IMPLEMENTATION OF A 3D EDUCATIONAL GAME FOR INDUSTRIAL ENGINEERS

By

David Bengoa-Terán

A project report submitted in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING
In
COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2013

Approved by:

José Borges, PhD Member, Graduate Committee	Date
Cristina Pomales-García, PhD Member, Graduate Committee	Date
Agustín Rullán, PhD Co-President, Graduate Committee	Date
Bienvenido Vélez, PhD President, Graduate Committee	Date
María de los A. Irizarry, PhD Graduate Studies Representative	Date
Pedro Rivera, PhD Chairperson of the Department	Date

ABSTRACT

Today's "Gamer Generation" has led to a rapid growth of the game industry, with a vast amount of money spent on commercial entertainment games, instead of educational games. One reason is because it is difficult to implement a game that accurately represents the concepts that one tries to teach while holding the students' attention.

There has been some research into the development of games which aim to teach science concepts, but not Industrial Engineering (IE) concepts. Therefore in order to address this issue, the purpose of this project is to develop a computer game, which is focused on exposing freshman IE students to fundamental concepts applicable to manufacturing systems and improve their problem-solving skills in complex unstructured problems. Besides being an educational game it can also be called a "serious video game", since it is classified as a "strategy video game". The player has to carefully plan and manage resources in order to win. The project scope will be a single-player video game for personal computers, with multi-platform support.

This game simulates daily activities in a factory in which users must make many decisions with the goal of fulfilling as many orders on time and as efficiently as possible. Some of the decisions that the user can control include: hiring or firing workers, buying or selling machines and equipment, increasing or decreasing storage space, setting unit loads for each transport activity, among others.

In order to perform this project implementation, Java has been used as the programming language, and the development has been done with Object Oriented programming. In addition, two synchronized databases engines have been used: SQLite as the local database and MySQL as the remote database. Finally, JMonkey has been used as the game engine; SimPack has been used as the discrete event engine; and some algorithm of Artificial Intelligence has been applied to determine the shortest way between two points in the factory.

RESUMEN

Actualmente la generación Gamer ha permitido un rápido crecimiento en la industria del juego, enfocándose en el desarrollo de juegos de entretenimiento, en vez de juegos educativos. Esto se debe a la complejidad que existe al desarrollar un juego educativo, ya que se debe conseguir que el jugador aprenda los conceptos que transmite el juego y a la vez, mantenga la atención de él.

Existen distintas investigaciones con el objetivo de implementar juegos educativos que enseñan diferentes materias pero muy poco se ha logrado para la Ingeniería Industrial. Es por esta razón que el presente proyecto pretende implementar un juego enfocado a los estudiantes recién ingresados de Ingeniería Industrial. Este juego además de ser educativo, también está clasificado como "juego de estrategia" porque el jugador estará planificando y administrando cuidadosamente los recursos con la finalidad de obtener la victoria. El producto obtenido es un juego para computadoras, con soporte multiplataforma, y para un solo jugador.

El juego consiste en la simulación de las actividades diarias de una fábrica, en donde el usuario toma decisiones con la finalidad de cumplir con el máximo pedidos que van llegando en el transcurso del juego. Entre las distintas decisiones se puede resaltar: contratar o despedir trabajadores, comprar o vender maquinarias y equipos de transporte, alquilar una mayor o menor cantidad de espacio en los almacenes y variar la cantidad de piezas o productos que se van a transportar entre las estaciones de trabajo.

Para llevar a cabo la implementación de este proyecto, se utilizó Java como lenguaje de programación, desarrollándose bajo una programación orientada a objetos. Además, se utilizó dos motores de base de datos que se encuentran sincronizados entre sí, uno local y otro remoto, SQLite y MySQL respectivamente. Finalmente, se está utilizando JMonkey como motor de juego, SimPack como generador de eventos discretos, y aplicando un algoritmo de Inteligencia Artificial para encontrar el camino más corto entre dos puntos en la fábrica.

To my grandma, parents, brothers and newborn niece.

Thanks for being there for me

ACKNOWLEDGEMENTS

First at all, I would like to thank to my advisor, Dr. Bienvenido Vélez, for his continue support and recommendation in every step of the game development. Also, I would like to thank my graduate committee, Dr. Agustín Rullán who trusted me from the beginning, gave me the opportunity to be part of his main research; Dr. Cristina Pomales-García who guided me in different stages of the game development, and promoted this achievement in every available event; and Dr. José Borges who provided me with his knowledge in the design patterns area and object oriented programming.

Thanks to my family, although they are far from me, I know that I always had their unconditional support. Also, I appreciate having met such good friends here in Puerto Rico, especially my Ecuadorian friends Ana Negrete and Marcelino Guachambala. Ana, I want to thank you for your help to complete this project report.

I want to thank to Yadrianna A., Christian V. and the tester team for their help the testing phase has been completed successfully. Also, Xiomara A. and Irving D. for their contribution helped in the game development.

Finally, I would like to acknowledge the support from the National Science Foundation, Award No. 0835990.

TABLE OF CONTENTS

ABSTRACT ii
RESUMEN iv
ACKNOWLEDGEMENTS vii
TABLES LIST xii
FIGURES LIST xiii
1. INTRODUCTION 1
1.1. Justification 1
1.2. Problem Statement 2
1.3. Our Approach 3
1.4. Contributions 4
1.5. Project Objectives 4
1.5.1. General Objective 5
1.5.2. Specific Objectives 5
1.6. Outline 6
2. RELATED WORK 7
2.1. How to Build Serious Games 7
2.2. Games for Science and Engineering Education 7
2.3. Pandora-Box project 8
2.4. Serious computer games as instructional technology 8
3. OVERVIEW OF THE GAMING EXPERIENCE 10
3.1. Introduction 10
3.2. Educational Concepts 10
3.2.1. Activities 10
3.2.2. Process Flowchart 11
3.2.3. Costs 12
3.2.4. Human Resources Management 14
3.2.5. Inventory Control 15
3.2.6. Production Operations 16
3.2.7. Probability Distributions and Timing 17
3.3. Initial Settings 18
3.4. Game Controls and Strategies 18

3.5. Game Goals 19
4. DEVELOPMENT PLATFORM 20
4.1. Java SE 6 20
4.2. JMonkeyEngine 3.0 SDK RC2 20
4.3. Nifty GUI 1.3.2 21
4.4. MySQL Database 21
4.5. SQLite Database 21
4.6. SimPack Discrete Event Engine 22
4.7. Artificial Intelligence – A* search algorithm 22
5. SOFTWARE REQUIREMENT SPECIFICATIONS 23
5.1. Introduction 23
5.1.1. Purpose 23
5.1.2. Project Scope 23
5.2. External Interface Requirements 24
5.2.1. User Interfaces 24
5.2.2. Hardware Interfaces 26
5.2.3. Software Interfaces 27
5.2.4. Communications Interface 28
5.3. Software Features 29
5.4. Design Constraints 32
5.5. Database Requirements 32
5.6. Software System Attributes 33
5.6.1. Reliability 33
5.6.2. Availability 33
5.6.3. Security 34
5.6.4. Maintainability 34
5.6.5. Portability 35
5.6.6. Performance 35
6. SOFTWARE DESIGN DESCRIPTION 36
6.1. System Architectural Design 36
6.1.1. System Overview 36
6.1.2. Presentation Layer 36

6.1.3. Business Layer 37
6.1.4. Data Access Layer 39
6.2. Game Activities based on Petri Nets Models 39
6.3. Package Diagram and Design Patterns 44
6.3.1. Package Diagram 44
6.3.2. Strategy Pattern 47
6.3.3. Façade Pattern 49
6.3.4. Singleton Pattern 51
6.4. Database Design 51
6.4.1. MySQL and SQLite Design 52
6.4.2. Databases Synchronization Process 54
6.5. Detailed Description of Components 55
6.5.1. Java Threads 55
6.5.2. A* Algorithm 56
6.5.3. SimPack 57
6.5.4. Nifty GUI 59
6.6. User Interface Design 59
6.6.1. Login and Main Menu 60
6.6.2. Main Information Panels 60
6.6.3. Control Screens 61
6.6.4. Activities Screens 64
6.6.5. Utilities Screens 66
6.6.6. Additional Screen 68
7. SOFTWARE INTEGRATION AND TESTING 69
7.1. System Overview 69
7.2. Test Plan 69
7.2.1. Features to be tested 70
7.2.2. Features not to be tested 70
7.2.3. Testing Tools and Environment 71
8. CONCLUSIONS AND FUTURE WORK 72
8.1. Conclusions 72
8.2. Future Works 72

8.2.1. Graphics Designs and Animation Improvements 73
8.2.2. More Advance Game Levels 73
8.2.3. Allow User to Create and Share New Levels 73
8.2.4. Game Migration to Android Devices 74
REFERENCES 75
APPENDIX A 78
APPENDIX B 90
GLOSSARY OF TERMS 92

TABLES LIST

Tables **Page**

Table 3-1: Requirements for each kind of activity 10
Table 3-2: Equations to obtain different times in each activity 17
Table 6-1: Package diagram detailed 45
Table 7-1: Features not tested 71

FIGURES LIST

Figures **Page**

Figure 3-1: Example of game process flowchart 12
Figure 3-2: Game overall – summary of expenditures and incomes 14
Figure 3-3: Hire/fire workers using general view (left) or specific view (right) 15
Figure 3-4: Raw Material Storage – three different parts 16
Figure 3-5: Assembly process 16
Figure 6-1: Overview of the game architecture 36
Figure 6-2: UML Sequence diagram – start a new game 38
Figure 6-3: Purchase activity: Petri Net (top) and Java state machine (bottom) 40
Figure 6-4: Operation activity: Petri Net (top) and Java state machine (bottom) 41
Figure 6-5: Transport activity: Petri Net (left) and Java state machine (right) 42
Figure 6-6: Management of activities in Java 43
Figure 6-7: Overall game – Package diagram 44
Figure 6-8: Strategy pattern – Class diagram 48
Figure 6-9: Façade pattern – Class diagram 50
Figure 6-10: Singleton pattern – Class diagram 51
Figure 6-11: Entity relationship diagram 53
Figure 6-12: Synchronization process between databases 55
Figure 6-13: A* pathfinding example in the game environment 57
Figure 6-15: GUI controls in the game environment 59
Figure 6-16: Login menu (left) and main menu (right) 60
Figure 6-17: Order, game log and overall screens merged in one screen 61
Figure 6-18: Allocate storage screen 61
Figure 6-19: Assign workers screen 62
Figure 6-20: Resources screen 63
Figure 6-21: Unit load screen 63
Figure 6-22: Operation activity screen 64
Figure 6-23: Purchase activity screen 65
Figure 6-24: Transport activity screen 65

Figure 6-25: Different stations: no-storage (left) and storage (right) 66
Figure 6-26: Machine screen 67
Figure 6-27: Operator screen 68
Figure 6-28: Game setup screen.. 68

1. INTRODUCTION

1.1. Justification

Today's “Gamer Generation”, so called because they grew up playing video games and spends hundreds and even thousands of hours during their most formative years playing video games [1]. This has led to a rapid growth of the game industry, with a vast amount of money spent on commercial entertainment games, instead of educational games. However, playing video games may have both positive and negative effects on cognition and behavior. As Douglas Gentile, an associate professor of psychology at Iowa State University [2] said “if content is chosen wisely, video games can actually enhance some skills.”

Concerning educational games, it is difficult to implement a game that accurately represents the concepts that one tries to teach while holding the students’ attention. There has been some research into the development of games which aim to teach science concepts, but not Industrial Engineering (IE) concepts. Therefore in order to address this issue, professor Agustín Rullán and collaborators proposed a research project entitled “Can gaming provide enough context to improve knowledge integration and retention in Engineering freshmen”, which is sponsored by the National Science Foundation with Award No. 0835990 and is described as a learning tool which “*...will be assessed in terms of improved retention, interest, and motivation of freshman IE students, as well as support of learning in context, improved student understanding of core concepts, and improved problem-solving skills in complex unstructured problems.*”

Once requirements were analyzed and the game rules were established, the purpose of this project is to create an educational video game based on a factory, to be played mainly by IE students and other people interested in learning IE concepts management. This game is classified as a “serious game” because the main purpose is to help freshman students learn about IE concepts. It could also be classified as a “strategy video game” because the player will carefully plan and manage resources with the objective of fulfilling orders on time and as efficiently as possible. This game is a simulation of the daily activities in a factory where decisions must be made to obtain desired results. Some of the decisions that the user can control include: number of active workers, machines, workstations, buckets, among others. The project scope will allow playing on personal computers, with different levels of difficulty. Although the graphic design of workers, machines and equipment has been simple, the objective is to help students learn various IE concepts.

1.2. Problem Statement

In order to achieve that freshmen industrial engineering (IE) students learn fundamental IE concepts applicable to manufacturing systems and improve problem-solving skills in complex unstructured problems, a serious educational game needs to be developed.

Previous research has demonstrated that science can be taught through games, however, industrial engineering is a field that is yet to be studied regarding the possibility of using games as a teaching tool.

To address this weakness, this project will focus on developing, deploying and assessing a video game based on a factory that aimed at freshmen students. This game will have features that will allow the student to discover and visualize several IE fundamental concepts, to acquire a notion about the types of decisions and actions that are required to run a manufacturing environment, to develop critical thinking skills and problem-solving skills, to acquire a vocabulary of IE terms, and to practice IE concepts to strengthen their technical background.

1.3. Our Approach

The original NSF proposal established the main activities of the game, which included: operation, transport, store, and purchase. These operations were described in detail using Petri Nets [3]. Our game is based on those activities but with some improvements. The new types of activities are: operation, transport, storage, purchase and shipping. Each level has multiple instances of these activities depending on the resources available. Furthermore, the set of activities, stations, storages, workers, machines, equipment, etc., are stored in two databases (remote and local) for each phase of the game. Therefore, when the administrator changes some data in the remote database, the players' instances will update their local databases. The player is able to change some features, demanding a cost for each one. For example: hiring or firing workers, buying or selling machines and equipment, allocating slots per storage station, among others. In addition, each purchase activity involves a cost, and the user's objective is to fulfill orders on time, as profitable as possible and avoiding bankruptcy.

Each activity requires a set of skills and each operator has a skill set. If an operator wants to perform the activity, this operator must have the required skills. Furthermore, the machines or equipment used can also break down after a certain time, so the game is considering a probabilistic time of repair. However, the user can avoid this break time, carrying out a preventive maintenance with its respective fee. Finally, normal, uniform and exponential distributions are used in order to simulate failures and repairs as realistically as possible.

1.4. Contributions

The game development lets us identify three (3) different approaches as contributions. First at all, the game is the principal tool of the main research, which aims to support learning of IE core concepts and improve problem-solving skills by freshman IE students. Once the game development is completed, then the main research can continue with its next process step.

Another contribution is developing a game that serves as a teaching tool for the Industrial Engineering area, given the lack of research through games. We expect it will encourage the development of other educational activities based on gaming.

Finally from a technical approach , we have successfully accomplished the use and integration of a game engine, an artificial intelligence algorithm, two different databases engines, a discrete event engine, and many more, with Java as a the language programming.

1.5. Project Objectives

1.5.1. General Objective

The main objective of this project is to develop a video game simulating a factory that will allow freshman students (players) to learn basic concepts of IE such as costs, human resources management, inventory control, production, planning, among others.

1.5.2. Specific Objectives

- A single-player video game made of different levels. Early levels will be easier than later levels.
- Each phase will have an initial configuration with a set of activities and a process flowchart, where the player has to figure out the changes in resource allocation to complete the arriving orders on time. Furthermore, these activities will have a given but changeable priority, and are dispatched for execution in priority order. All the information required for each phase will be stored in a database.
- Each phase will have available a set of workers, machines, equipment and storage areas. The game will allow the users to hire or fire workers, and buy or sell machines or equipment, to improve savings or reduce expenditures.
- The objective for each phase will be to dispatch a given set of orders with maximal profitability. Each order will be described by the type of part, quantity, and due date. Orders arrive as the game progresses.
- Users will be able to change some controls while the game is running, such as the "unit load" for each transport activity, the "reorder point", "order quantity" and "supplier" for the purchase activity. Also, the player will have more options in the "settings section."

- It is important that the game allows to “play” and “pause” an ongoing game, and change the “time factor” between 1/16x, 1/8x, 1/4x, 1/2x, 1x, 2x, 4x, and 8x, so that users have time to analyze their decisions.
- Finally, the game will allow the user to know the overall state of the game including: the available cash, i.e. initial capital minus expenditures plus incomes; total expenditures, e.g. overhead cost per hour, workers’ salaries, machines cost per hour, storage cost per hour, among others; total incomes, e.g. sales of machines, equipment and parts; and total profit, i.e. incomes minus expenditures.

1.6. Outline

The next chapters of this document are organized as follows: Chapter 2 describes related works in the area of educational games. Chapter 3 describes the most important information about the factory features. Chapter 4 describes technologies used in this application. Chapter 5 relates the software requirement specifications. Chapter 6 provides detailed information about software design description, based in [5]. Chapter 7 software integration and testing, and is based in [6], [7] and [8]. Finally, Chapter 8 presents conclusions and recommendations.

2. RELATED WORK

This section presents related works about educational games regarding the immune system and training of crisis managers. Also, other studies about the use of games as a teaching tool in the school, university or related.

2.1. How to Build Serious Games

The computer game "Immune Attack" was created by a team of computer programmers as a serious science-based game [17]. This game combined a realistic 3D depiction of biological structure and function of the human body with educational technologies to teach immunology to high school and college freshmen. As part of this experience, the developers describe the process of creating a serious game as a challenging endeavor, as it has to satisfy experts and novices while addressing deeply held pedagogical assumptions, distinct expert viewpoints, integration of gameplay and learning content, among others.

2.2. Games for Science and Engineering Education

The United States has a relatively small percentage of engineering graduates compared to other developing nations, a statistic that is attributed to the perception that science and engineering is a boring course of study. The author suggests that this perception can be changed if computer programs are used to teach science and engineering topics, from kindergarten through grade twelve, encouraging more students to continue college studies in these fields [18].

2.3. Pandora-Box project

Crisis management prevents emergency situations from turning into disasters and training plays an important role in preparing the crisis manager. In order to achieve this, the Pandora project [19] assists crisis managers' training using innovative technology. In particular, it is creating a tool that collaborates with traditional training methods to generate and to improve decision-making skills for trainees. It shows three important aspects: (a) a novel use of timeline based planning as the core element in a dynamic training environment for crisis managers; (b) a continuous loop of planning, execution, and plan adaptation is created to support personalized training; (c) a trainer is provided with a set of functionalities that allow him/her to maintain and adapt a "lesson plan" as the basis for the interaction between him/her and the involved trainees.

2.4. Serious computer games as instructional technology

The potential value of serious computer games for learning seems high, but there is still some degree of resistance to the use of games in a classroom. A reasonable way to convince teachers to use games as a teaching tool is through pedagogy, connecting elements of existing game designs with accepted learning and instructional theories. At the Faculty of Education of the University of Ljubljana, the serious game TimeMesh [20] has been developed in the framework of the Comenius programme. The game is intended to be used for learning history in primary and secondary schools, but at the same time students learn about different cultures and social relations in Europe in different historical periods. Research results show increasing students' motivation and their interest for topics covered by the game. On the other hand, some teachers do not completely agree to use serious games because games can be too time-consuming for use in a classroom; however, they are willing to present the games as a home-based learning activity.

3. OVERVIEW OF THE GAMING EXPERIENCE

3.1. Introduction

The main objective of this chapter is to explain the game development from a different technical approach. This chapter denotes different IE concepts that the user should learn or notice during the game, initial settings for each game level, controls available for the user, and game goals. It outlines the different activities during the game, the meaning of the process flowchart, different costs involved, human resource management, inventory control in the factory, production operations, and the use of probability distributions.

3.2. Educational Concepts

3.2.1. Activities

The game summarizes the daily activities of the factory in four (4) main activities: purchase, transport, operation, and shipping. These activities are executed constantly during the game, and depending on the activities the requirements vary, as shown in Table 3-1. The number for each kind of activity varies depending on the level. Also, early levels will be easier than later levels which have more activities.

Table 3-1: Requirements for each kind of activity

Activity	Validation Process – Step by Step
Purchase	1. $PartInventory + UnitsToArrive \leq OrderPoint$ 2. $StationInventory + UnitsToArrive + OrderQuantity \leq StationCapacity$ 3. $OrderQuantity \times PartPrice \leq CurrentMoney$
Transport	1. Available qualified operator & available transport equipment 2. $EndStationInventory + UnitsToArrive + PartsToMove \leq$

	$EndStationCapacity$ 3. $InitialStationInventory - UnitsToRemove \geq PartsToMove$ 4. Available slot in initial/end station, if required
Operation	1. Available qualified operator & available machine 2. $StationInventory + UnitsToArrive + PartsToProduce \leq StationCapacity$ 3. $PartRequired \times QuantityRequired \leq PartInventory - PartsToRemove$
Shipping	1. Next order required 2. $StationInventory \geq OrderQuantityRequired$

The game allows players to change parameters for some activities in order to get a desired result. Purchase activity allows changing the reorder point, order quantity, and supplier. Transport activity allows changing the amount of parts to be transported, or unit load. Other activities will not allow changing any parameter, for example the operation activity will produce the same amount of parts that have been set, and the shipping activity will ship the number of parts required in the waiting order. In the operation activity the player can determine the number of machines available as well as number of operators and their respective skills.

3.2.2. Process Flowchart

Figure 3-1 represents the set of activities that occur in the factory using a process flow symbols. It allows the player to get a general understanding of the game, in particular the flow of materials. Each game level has a particular process flowchart. For example Figure 3-1 shows the hardest level with multiple transport activities and assembly processes.

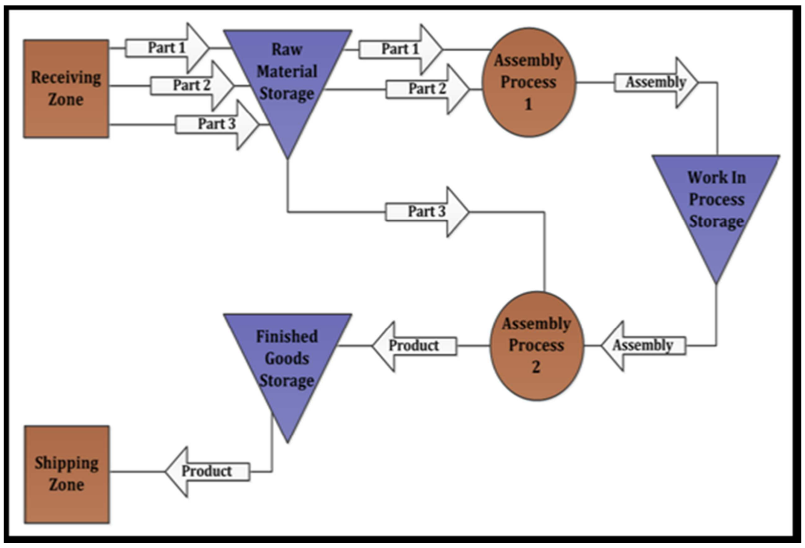


Figure 3-1: Example of game process flowchart

The squares represent stations; triangles denote the different type of storages (i.e. raw material, work in process, and finished goods); and circles represent the assembly and cutting process. In the Figure 3-1, three different parts are moved from the *receiving station* to the *raw material storage*. Then two parts are moved to the *first assembly station*, to create one intermediate part that is stored in the *work in process storage*. After that, this part and the part three are moved to the *second assembly station* to create the final product that is stored in the *finished goods storage*. Finally, once an order is received, this product is moved to the *dispatch station*.

3.2.3. Costs

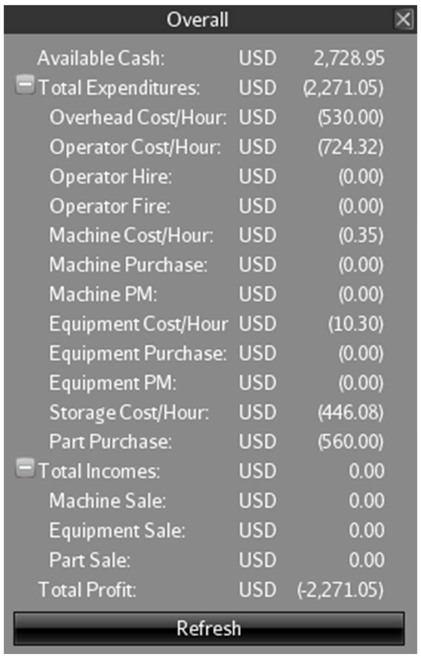
This game manages different costs in order to mimic the realities in a factory. Figure 3-2 shows the summary of expenditures and incomes, and the costs include:

- Storage stations. It includes raw material, work in process and finished goods station. Each of them has a limited number of slots, depending of their station size.

The player will choose the number of available slots and will pay a fee per hour even if they are not in use.

- Workers. Once the game starts, the player will choose the number of workers to hire and will also be able to fire some or all of them. Both processes will incur in costs, meaning that the player will spend money hiring and firing workers. Workers will be paid an hourly salary which will depend on the worker’s skills, including: Material Handler, Machine Operator, and Versatile.
- Machine and equipment. The game allows the player to buy and sell the available machines and equipment as the player wishes. Also, a cost per hour will be charged for machine and equipment usage, and after a certain time these will require preventive maintenance, otherwise they will break down. This maintenance implies another cost, so the player will choose between paying this cost, continue working, or wait a few minutes for automatic repair. Moreover, the machine or equipment will have a depreciation cost that will impact the sell price.
- Supplier. The game will provide the player a list of suppliers to be selected to start the process, the first step is to purchase raw material, as shown in Figure 3-1 above. The cost of the raw material will depend on the selected supplier and the order quantity. Each supplier’s catalog for a specific part provides three (3) different prices based on time to receive the order.
- Overhead. This is an average cost per hour that includes an ongoing expense of operating the factory, for example: gas, electricity, taxes, telephone bills, other wages, etc.

- Part. Product sale is the last process and does not qualify as a cost, as shown in Figure 3-2 below. It is important because without this step the factory will go bankrupt as soon as the costs are higher than the initial capital.

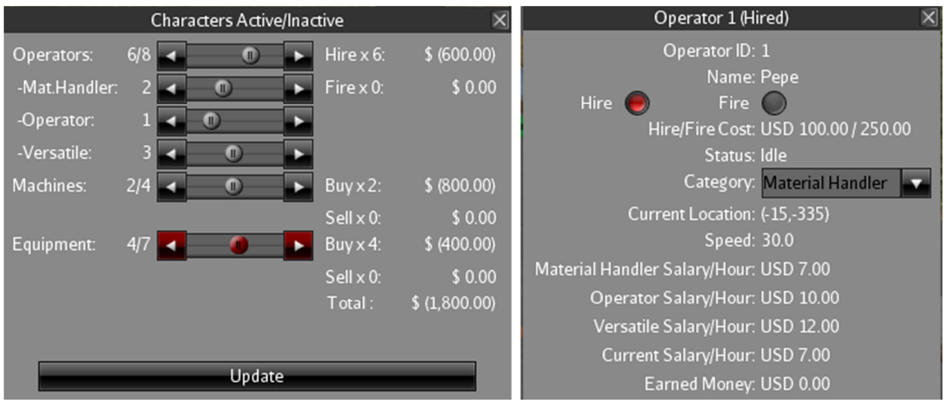


Overall		
Available Cash:	USD	2,728.95
⊟ Total Expenditures:	USD	(2,271.05)
Overhead Cost/Hour:	USD	(530.00)
Operator Cost/Hour:	USD	(724.32)
Operator Hire:	USD	(0.00)
Operator Fire:	USD	(0.00)
Machine Cost/Hour:	USD	(0.35)
Machine Purchase:	USD	(0.00)
Machine PM:	USD	(0.00)
Equipment Cost/Hour	USD	(10.30)
Equipment Purchase:	USD	(0.00)
Equipment PM:	USD	(0.00)
Storage Cost/Hour:	USD	(446.08)
Part Purchase:	USD	(560.00)
⊟ Total Incomes:	USD	0.00
Machine Sale:	USD	0.00
Equipment Sale:	USD	0.00
Part Sale:	USD	0.00
Total Profit:	USD	(-2,271.05)

Figure 3-2: Game overall – summary of expenditures and incomes

3.2.4. Human Resources Management

The management of the workers is an important factor in the game. Hiring or firing workers increases or decreases the production of finished goods, and therefore the fulfillment of the arriving orders. In addition, hiring workers involves choosing a role type for each worker. As we mentioned above, there are three (3) types: Material Handler, who uses equipment (i.e. hand trolleys) to transport parts and products in the factory; operators, who work directly with production; and versatile workers that can perform both roles. Also, it is important for the player to control how many workers of each type to have since it has a direct impact on productivity and increases costs. Figure 3-3 shows the two ways to hire and fire workers, and deciding the worker’s role.


Characters Active/Inactive

Operators:	6/8	Hire x 6:	$ (600.00)
-Mat.Handler:	2	Fire x 0:	$ 0.00
-Operator:	1		
-Versatile:	3		
Machines:	2/4	Buy x 2:	$ (800.00)
		Sell x 0:	$ 0.00
Equipment:	4/7	Buy x 4:	$ (400.00)
		Sell x 0:	$ 0.00
		Total :	$ (1,800.00)

Update

Operator 1 (Hired)

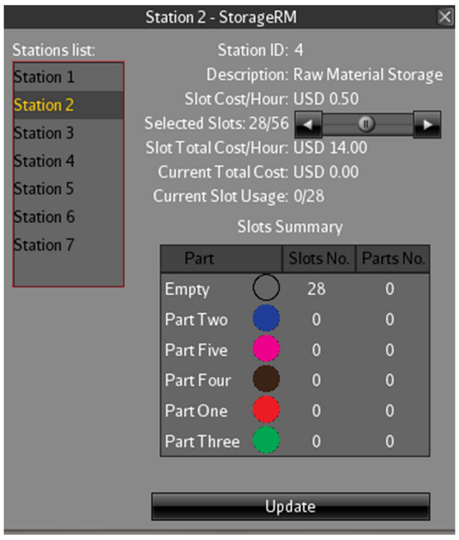
Operator ID:	1
Name:	Pepe
Hire ●	Fire ○
Hire/Fire Cost:	USD 100.00 / 250.00
Status:	Idle
Category:	Material Handler
Current Location:	(-15,-335)
Speed:	30.0
Material Handler Salary/Hour:	USD 7.00
Operator Salary/Hour:	USD 10.00
Versatile Salary/Hour:	USD 12.00
Current Salary/Hour:	USD 7.00
Earned Money:	USD 0.00

Figure 3-3: Hire/fire workers using general view (left) or specific view (right)

3.2.5. Inventory Control

The ease of use of the game allows the player to know at every moment the exact amount of inventory for each part or product. This game feature is available to the player as a general inventory for the part, which can be found in the Part window, or an inventory for a specific station, or it can be found in the receiving dock located in the receiving zone. In addition, the inventory in a specific storage station (i.e. raw material, work in process, finished goods) is different from the receiving dock, because the storage stations are controlled through slots available and is not limited by part types, as shown in Figure 3-4, as opposed to the receiving dock which limits capacity and type of part.

However, the effective management of inventory control requires an adequate knowledge of inventories for different types of activities. For example, the reorder point and order quantity for purchase activity, the unit load for transport activity, the amount of required parts for the operation activity, and the amount of finished goods to fulfill the orders for shipping activity.



Station 2 - StorageRM

Stations list:

- Station 1
- Station 2
- Station 3
- Station 4
- Station 5
- Station 6
- Station 7

Station ID: 4
Description: Raw Material Storage
Slot Cost/Hour: USD 0.50
Selected Slots: 28/56
Slot Total Cost/Hour: USD 14.00
Current Total Cost: USD 0.00
Current Slot Usage: 0/28

Slots Summary

Part		Slots No.	Parts No.
Empty	○	28	0
Part Two	●	0	0
Part Five	●	0	0
Part Four	●	0	0
Part One	●	0	0
Part Three	●	0	0

Update

Figure 3-4: Raw Material Storage – three different parts

3.2.6. Production Operations

Currently the game supports two kinds of operations: cutting and assembly. Both processes (example shown in Figure 3-5) can be found in different game levels. Each process produces as many parts as it is specified in the operation activity and requires an adequate machine and a skilled operator.

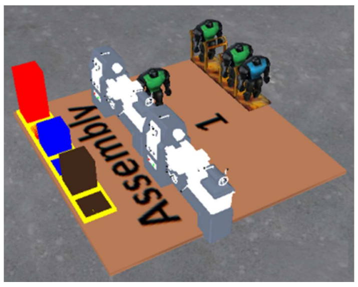


Figure 3-5: Assembly process

3.2.7. Probability Distributions and Timing

The use of statistical distributions is important in the game development process, in order to achieve results that are as realistic as possible. Uniform and exponential distributions are used as part of some elements in the factory, e.g. machine time between failures, equipment repair times, and order shipments. Moreover, these statistical distributions allow calculating different equations in order to obtain a specific time to execute some steps of many activities. Finally, other equations only require constant values to get different times for each execution. Table 3-2 shows the detailed equations.

Table 3-2: Equations to obtain different times in each activity

Activity	Sub-Activity	Equation
Operation	Machine time	$\text{Time} = \text{quantity} \times \text{factorMachine} \times \text{timeToCompleteTask}$
Transport	Time between failures	$\text{Time} = \text{Exponential}(x)$
	Repair time	$\text{Time} = \text{Uniform}(x, y)$
	Pick up / Placement time	$\text{Time} = \text{quantity} \times \text{timeToCompleteTask} \times \text{factorOperator}$
Operation and Transport	Walk alone	$\text{Time} = \left(\frac{\text{distance}}{\text{speed}}\right) \times \text{factorOperator}$
Transport	Walk loaded/unloaded with equipment	$\text{Time} = \left(\frac{\text{distance}}{\text{speed}}\right) \times \text{factorOperator} \times \text{factorEquipment}$
Shipping	Time to ship the order	$\text{Time} = \text{Uniform}(x, y)$
		$\text{speed} = \text{constant}$ $\text{timeToCompleteTask} = \text{constant}$ $\text{factorOperator} = \text{Uniform}(x, y)$ $\text{factorEquipment} = \text{Uniform}(x, y)$ $\text{factorMachine} = \text{Uniform}(x, y)$

3.3. Initial Settings

The initial settings process is comprised of six (6) different steps, where the first four are required and the last two are optional, as shown in Figure 6-23. During the setting process, the user must know that some steps incur in costs, for example step 1, 2, and 4. In the first step, the user will hire some workers, and will buy some machines and equipment. Also, he/she will need to choose the role for each worker. The second step allows allocating a quantity of slots for storages. In case the slots are filled, then the game will not be able to store more parts until some slots are released. The third step indicates the quantity of parts or products to be moved from one station to another. The fourth step denotes the reorder point and the order quantity of raw material to the next purchase. Optional steps are not necessary to be setup, given that the game can be played without them. Fifth step allows the user to assign one or more workers to one or more activities. The last step allows modifying the execution priority for each type of activity.

Once the player has started a new game, he or she will need to setup some features of the game to start playing it. This game release provides two (2) ways to adjust settings: (1) the "default setup" button or automatic configuration, as shown in Figure 6-23, lets novice players to focus on the game instead of setup the features; (2) the manual configuration allows expert users or user with some game knowledge to setup the game features as they desire.

3.4. Game Controls and Strategies

The game provides a set of controls that allow users to manage the game in different ways, and to obtain many results as possible. Some controls are implemented to execute several tasks at same time but giving fewer details about each task. Others give more information and execute each task independently. Figure 3-3 shows the “Resource” screen at the left, which allows the user to hire/fire workers and buy/sell machines and equipments, and the “Operator” screen at the right, which shows detailed information of the operator and allows hiring or firing this current operator.

In accordance with the wide range of options to control the game, the user can implement different strategies to obtain diverse results. One strategy can be to use the least amount of workers, machine and equipment. This strategy may not generate much profit but could win the game. However, another strategy could be fulfilling every order as possible, regardless the quantity of workers hired, and the amount of machines and equipment purchased. Therefore, the more the user plays, the more strategies the user could implement.

3.5. Game Goals

The main goal to win the game is to fulfill as many orders as are required without going to bankruptcy, where the quantity of orders varies depending of the game level. Another goal is to earn the most profit as possible with the least usage of workers, machines and equipment. A final objective but no less important is learning Industrial Engineering concepts through play, given that, it is an educational game.

4. DEVELOPMENT PLATFORM

The project aims to provide a video game that is extensible, portable, and efficient. In order to achieve these goals, open-source software was used for the game engine, discrete event engine, and database.

4.1. Java SE 6

Java [9] is used as the programming language due to the features and facilities it provides to support Object-Oriented programming. Portability is one of the features in Java which does not require that the program be developed using Windows OS. In order to take advantage of this feature, the game as a product is able to be executed in every platform, e.g. Mac OS, Linux and Windows.

4.2. JMonkeyEngine 3.0 SDK RC2

This is the current version of the game engine. It was rewritten from the ground up to accommodate modern standards in game development. This new version is only 2 years old, but it reuses the best pieces of code and best practice from many generations of Java game programming [10].

However, currently there are many game engines, including commercial and freeware. For example, Unity [24] has both versions and allows creating game for web, mobile and game consoles. UDK (Unreal Development Kit) [25] is freeware. GameStart 2D/3D [26] is freeware, allows creating multi-platform games, and is based on C++. Cocos2d [27] is an open source 2D game framework and multi-platform. However, before starting the game development, it is important to know something about each game engine available, given that each one has pros and cons. Therefore, once you know the category of your game, then you should decide the game engine.

4.3. Nifty GUI 1.3.2

This Java library creates interactive user interfaces and is well integrated into many existing rendering systems (JME3, JME2, LWJGL, JOGL, Slick2D and even Java2D). The actual GUI is stored in XML files or it can be created directly from Java. Java is used to respond to events generated by the GUI and to modify the GUI to reflect changes in the state of your application. Additionally there is a large set of effects available that can be used to modify the appearance of the GUI. [11]

4.4. MySQL Database

MySQL is the world’s most popular open source relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases. It provides high scalability and performance. There is also a free MySQL Workbench (GUI Tool) available, that can be used to efficiently design, manage, and document database schemata [12]. This database is used in the game as the remote database.

4.5. SQLite Database

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world [13]. In the project, this database is used as the local database, which means each game instance will contain SQLite database as an extra file but it will act as an engine.

4.6. SimPack Discrete Event Engine

SimPack is a collection of routines and programs for computer simulation developed by Dr. Paul Fishwick [15] from the Department of Computer and Information Science and Engineering at the University of Florida. The source of SimPack is Open Source under a Gnu Public License (GPL). It has a version in C++ but also a later version in Java which is the one that was finally adopted for the game. The decision was made because it would be easier to develop the code using the selected game engine, JMonkey, which provided the opportunity to be OS platform-independent since it is written in Java.

4.7. Artificial Intelligence – A* search algorithm

It is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently traversable path between points, called nodes. Noted for its performance and accuracy, it is widely used. A* uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals) [16]. In the project, it is used to find the path (set of pixels) of the transport activity, from one station to another.

5. SOFTWARE REQUIREMENT SPECIFICATIONS

5.1. Introduction

5.1.1. Purpose

The purpose of this chapter is to give a complete description of the Factory Game. It will explain the features, interfaces, performance and development requirements of this game. The chapter structure shown below is based on the IEEE standards [4]. It is supplemented by the software design description and the software integration and testing chapters.

5.1.2. Project Scope

This game is a cross-platform application that allows a single player to improve his/her understanding of some IE core concepts and other problem-solving skills in complex unstructured problems. This software release is available only for personal and desktop computers. It provides the user automatic updates, managed through different synchronized databases, i.e. a remote database in a server and a local database for each user. The game has been built with a 3D game engine and contains several controls (i.e. game variables) to achieve different results. In general, it focuses on providing an easy to play, and a realistic playing experience.

5.2. External Interface Requirements

5.2.1. User Interfaces

5.2.1.1. Login and Create New User

The login process is the introductory screen in the game to further use other features. In case the player does not have a “registered user”, he/she must go to the screen “create a new user”. This screen should be filled thoroughly and requires some user information, which is essential to know the player`s profile. Also, the password is created automatically and sent to the email registered. This last process will also validate the authenticity of the email. Screens are implemented using Nifty GUI controls.

5.2.1.2. Main Menu

The game provides the user a list of principal options en el main menu, e.g. new game, profile, options, user manual, credits, switch user and quit game. Once the player clicks on “new game”, it will change to a new screen showing the list of games available, with appropriate information about the status of each game level, scores, number of attempts, description and others. The “profile” button will redirect to another screen, allowing to the player to change his or her user profile. The “options” button will open a new screen, where it allows the player to adjust setting to their specifications: “sound” option will allow enabling and disabling the sound; “controls” option will allow changing the hot keys established; finally “screen” option will enable the user to change the screen size resolution and full-screen resolution. The “user manual” button will open a web browser, loading the user manual with definitions of the game and examples to learn it. The “credits” button will show information about the team who developed the game, also referencing to the NSF support. These screens are implemented using Nifty GUI controls.

5.2.1.3. Game Environment and Controls

The main user interface is the game environment, given that it will show the factory with 3D graphics design, moving every graphic element in accordance to the game activities, and with several controls, allowing the player to take control of the game. This development is integrated with the Nifty GUI and the JMonkey engine which means that the user will interact with both. For example: the actions taken by Nifty GUI controls are reflected in the game environment (i.e. JMonkey engine). The 3D graphic design shown in the game environment was designed by the art team, headed by Professor Felix A. Zapata.

Game development is not only a set of visual objects and controls; it requires a set of sounds that should be played in different moments during the game. For example, the game will have a background music that can be disabled; some activities like “processing a new part” will have an assonantal sound according the factory machine used; equipment breaks will have another sound, and so on. Most sounds used in the game have been found free on the Internet, except of the background music, which was provided by the music team.

5.2.1.4. Pop-up Windows

The use of pop-ups in the game has been applied as shown below:

- Starting the game. Once the game starts it will show a pop-up that will synchronize the game data between the remote and local databases.

- Login user. Once the user is logged into the game, it will show a pop-up that will synchronize the user data between both databases.
- Quit game. When the user clicks on the "quit game" button, the game will show a warning before the game closes.
- Win/Lost game. Even if, the user wins or lost a game, it will show a pop-up with the game statistics and some options like: "restart", "next game", "quit game", among others.

5.2.2. Hardware Interfaces

5.2.2.1. Multi-platforms

This game version is a desktop application, designed to be played correctly in different platforms, for example:

- Windows: version XP, 7 and 8.
- Mac OS: version Lion and Mountain Lion
- Linux: different distributions, e.g. Ubuntu, Mint, and Fedora.

5.2.2.2. Graphic Library

The game development uses OpenGL library [14] to be able to show different graphics animations and 3D designs, and also because it supports multi-platform API for rendering 2D and 3D computer graphics, that is used to interact with a GPU, to achieve hardware-accelerated rendering. Currently, the library is in the version 4.3 but in order to run this game in computers with standard and old graphic cards, the game should use the version 1.1.

5.2.2.3. Computer Devices

Playing this game requires a mouse and a keyboard. The mouse device will allow the player to move in the game environment, click on several game objects, and click on different menu buttons. The keyboard device enables the user mainly to two main objectives: game login and for the use of hot keys to move the game camera. Other computer devices are not allowed in this game.

5.2.3. Software Interfaces

5.2.3.1. Databases

The game uses two different relational databases for an appropriate management of the game and user data. In the user machine, one game instance stores the gaming application data and user data in a local database, using the SQLite relational database. The connection with this database is done by a single connection string, using a singleton design pattern [21] that aims to decrease the excessive use of memory. This connection must be fully configurable, so it could be updated remotely.

Moreover MySQL database is installed in a server machine to collect user data and to store the latest version of each gaming level. This database instance could be managed through the MySQL Workbench 5.2 application or MySQL Command Line application by the user administrator.

5.2.3.2. Main Libraries

Different libraries are used as part of game development. The discrete event machine called SimPack is used to schedule all activities during the game. The Nifty GUI library will allow the player to control the game through several graphical controls. The use of A* algorithm [16], that manages an internal matrix modeling the game environment, provides the game engine the path on route between any two points in the factory. This algorithm is executed only when is necessary, given that it takes too many resources like memory RAM and CPU clock cycles.

5.2.4. Communications Interface

5.2.4.1. Database Synchronization

The synchronization process between many SQLite database instances (i.e. client side) and one MySQL instance (i.e. server side) will take different steps. First, it starts each time the player executes the application, so the client side must have an access key (i.e. user and password). Second, the server will have the firewall port access open but only will allow secured and verified access. Third and finally, the access in the database must be restricted and limited, allowing only the execution of some queries, e.g. select, update, and insert statements; and modifying data of its current user.

5.2.4.2. Communication Protocols

The process of registering a new user will involve sending an e-mail with the new password generated. In order to achieve that, SMTP will be used with previous settings stored in an encrypted file located in the player machine. For future changes, it could be updated through the database sync process or with a new game version.

The database synchronization process mentioned above will use the TCP/IP protocol and default port number 3306, which are the common ones in MySQL database. Also, this information will be stored in an encrypted file for each player machine.

5.3. Software Features

This stage in the software development life cycle has been a challenging work for the team. The features have been adding, reducing, and changing constantly during this whole process, given the lack of experience developing games but fortunately the software specifications have been established, and these are:

- **Synchronization process.** This feature will be executed through different pop-ups as mentioned in Section 4.2.1.4 (i.e. starting the game and login user). Also, once the user finishes one game level, the game engine should trigger this sync process as mentioned in Section 4.5. More details about the synchronization process, can be found in Section 4.2.4.1.
- **User Profile.** A new user will be required to fill the registration screen as mentioned in the Section 4.2.1.1. This screen contains significant fields in order to use this user information for future studies. These fields will be: name, last name, gender, status (i.e. undergraduate student, graduate student, corporate, among others), degree, country and email. Once the new player completes it, he or she should receive a message with the new password in the email account.
- **Main Menu.** This feature will be a screen that centralizes the different ways that a player can move in the game. The "New Game" button should be the most valuable

option to the player. For more details about this menu and the game list could be found in Section 4.2.1.2.

- **Game Environment.** This feature is the most notable graphically because the environment is based on a factory, and each game level will be played in that environment. Moreover, the factory should show every operator (i.e. material handler, line operator, and versatile), every machine and equipment, and all the different parts that are enrolled for each game level. Finally, every object contained in the environment should be animated according to actions that the user is taken.
- **GUI Controls.** This set of controls allows the user to take the control of the game. It means if the user sets up controls in some way, the game environment will show graphics changes and statistical changes (i.e. affecting costs, fulfilling or missing orders, among others). The list of the controls are:
    - Play and pause button.
    - Time slider. The control should increase and decrease the game speed.
    - Clock for the game, and a timer for "next order due" and "next purchase".
    - Game setup screen. Required configuration before starting the game
    - Overall screen. It should show costs detailed.
    - Order screen. It should show arriving orders with its details
    - Game log screen. Every notable activity is shown in this screen
    - List of main buttons:
        - Control option
            - **Allocate storages screen**. Can be used to allocate an amount of slots available for each stock. It should entail in costs.

    - **Assign operators screen**. Can be used to assign operators to one or more activities, depending of the role required.
    - **Priority activities screen**. Can be used to prioritize the sequence of activities execution.
    - **Resources screen**. Can be used to hire and fire operators, and to choose the function for each one. Also, to buy and sell machines and equipment.
    - **Unit load screen**. Can be used to set the quantity of parts to be moved during each transport activity.
- Activities option
    - **Purchase screen**. Can be used to set the reorder point and order quantity, and to choose the desired supplier.
    - **Operation screen**. Can be used to change the quantity of parts to assemble or process.
    - **Transport screen**. Can be used to set the quantity of parts to move during one transport activity.
- Utilities option
    - **Station screen**. Only stations qualified as storages will allow changing the quantity of storage slots available. In other type of stations this screen should be read-only.
    - **Machine and Equipment screen**. Can be used to buy, sell machines, or to make preventive maintenances, which incurs in additional costs. The screen also shows some critical information

such as percentage of depreciation, cost per hour, percentage of availability, percentage of usage, among others.

- **Operator screen**. Can be used to hire and fire operators, and to update the functions carried out by the operators.
- Part screen. This screen is read-only and shows critical information about the part such as type of unit, current stock, price for sale, and assembly parts required.
- Supplier screen. This screen is read-only and shows information about the supplier and its catalogue of parts with different prices. These prices are according to the quantity of items required by the factory.

5.4. Design Constraints

In accordance that this is an educational game, and its target audience is Industrial Engineering freshman students, the wording is essential in the design of user interfaces. For example, labels and messages shown in different screens should use technical words.

For this current game version, the design of user interfaces will use the default style that comes in the game engine. However, in a future release, the design should be customized and the game should allow the user to set it up.

5.5. Database Requirements

The communication with the remote database must be done only per request, which means: (1) open connection, (2) perform required query, and (3) close connection. It should not keep logged with MySQL, and connections should be done only for synchronization purposes.

In user machine, the connection with SQLite will be done at the following instances: (1) when the player starts the application, it will load the list of game levels for the current player. (2) When the player clicks on the button of “Start Game”, then the engine will gather all data required for this game level. Finally (3) it will happen each time that the game level finishes, winning or losing and the game engine should update the corresponding user records.

5.6. Software System Attributes

5.6.1. Reliability

To ensure reliability in the game, the user data stored in the remote database must be of real players. This is accomplished during the process of creating a new user, given that the game will require an email, which will be validated through sending a message to this email, attaching the new generated password by the game engine. Once the player receives this password to his or her email account, he or she will be able to start playing. Furthermore, the user account automatically should be activated the first time the user logs in. On the other side, the game server will validate every few days that if some user account has not been activated for some time, it should be dropped.

5.6.2. Availability

Once the player has downloaded and installed the game, the next step is to create a new user account. This process will require an Internet connection because the software should synchronize the user data provided to avoid duplicated user names. So the game server should be available at all times. Otherwise, none new user will be allowed to play the game.

Furthermore, other steps in the synchronization process will include game data and user data synchronization. If the remote server is not available, it will not be able to get the current user data, and users will not have new levels and/or updated levels.

5.6.3. Security

The security in the game was considered in different stages. When someone tries to play, the game should require a username and password which should be registered in the remote database. Therefore only registered users will be allowed playing, thus ensuring validated access to the game.

Additionally, the user data gotten in the process of “creating a new user” and through the game should be serialized and encrypted before sending to the remote database. Therefore, the game assures protection data of every user. Furthermore, when some game instance tries to access the remote database, it should be with a username and password different than the current user keys, which should be provided in the game development process.

5.6.4. Maintainability

Currently, this game version is the first release available to everybody, but continuously it will have further versions with improvements done. For future releases, it should be based on fixed bugs, provide better graphics objects (i.e. animations and design), and improve game performance.

Moreover, on the server side, it is managed by a user administrator who will be in charge of incorporating new game levels, updating current game levels, collecting statistical data users, among others. Finally, this user should update the new installer of the game where it is hosted.

5.6.5. Portability

In order to allow the game to be played in different platforms (i.e. Windows, Mac OS, and Linux) it has been developed using the Java language programming. Java also facilitates creating an installer for each platform.

5.6.6. Performance

Given that currently most of the new computers have powerful components, as graphic cards, memories, and processors, game performance should not be a major issue. However, there are still computers with old components that will not allow the game to run, or difficulties may occur making the game run slower. The game should be continuously improved to make the experience enjoyable, regardless of the hardware.

6. SOFTWARE DESIGN DESCRIPTION

6.1. System Architectural Design

6.1.1. System Overview

The development of the proposed game will follow the system architecture shown in Figure 6-1. This architecture is comprised of four main tiers (layers) each having specific functions. (1) The presentation layer; provides the graphical user interface (GUI). (2) The business layer; which contains and controls the artificial intelligence algorithm, the discrete event engine, input manager, sound engine, graphic engine, and game engine. (3) The data access layer; provides simplified access to data stored in persistent storage. (4) The data source; stores the set of tables and routines, the game logic and the user data.

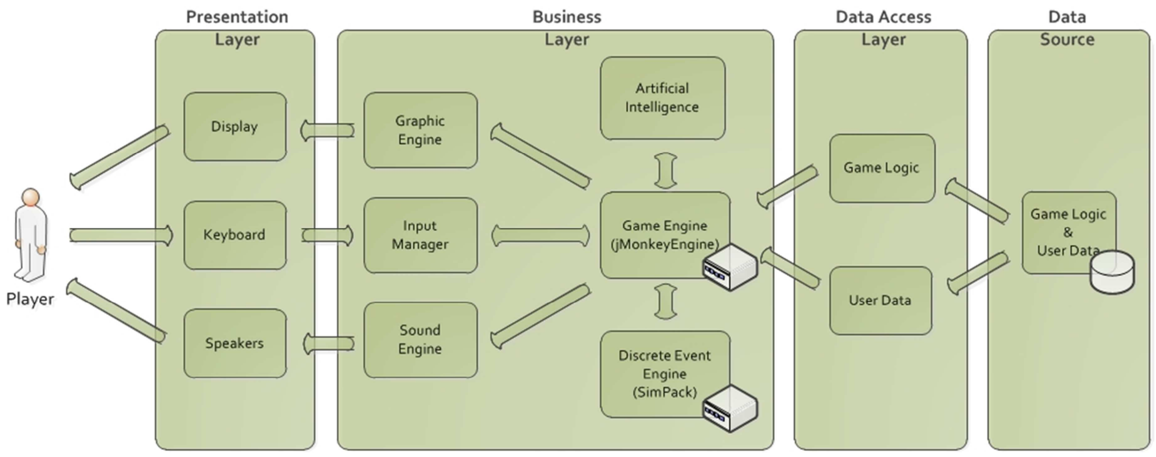


Figure 6-1: Overview of the game architecture

6.1.2. Presentation Layer

This is the top layer in the architecture, and is the one the user will interact with. On this layer the user will log into the application, select the game that will be played, and begin playing the first levels of the game. Additionally, the user will be able to complete the initial configuration of the game or choose a default configuration.

Representation layer uses the Nifty GUI library. This Java library provides a set of tools to implement each visible element that the player will use (e.g. window, panel, text field, label, dropdown, list and other controls.) Sometimes the controls available on this library are not enough to give the user the flexibility desired. Since the library is open source new tools can be incorporated to achieve an improved new experience.

The presentation layer also incorporates the graphic elements that make up the gaming stage such as: world, workshop, stations, machines, equipment, parts, buckets, operators and their associated animations. The presentation layer is mostly managed by the JMonkey game engine [10].

6.1.3. Business Layer

The business layer is the most important one because it contains the core elements of the game. It serves as the interface between the presentation layer and the data layer. The presentation layer processes the user actions (mouse and/or keyboard events), and sends these events to the game engine through the input manager. Depending on the action, the game engine will call the data layer to carry out the corresponding process. For example, Figure 6-2 depicts a UML sequence diagram [22] with the process required to start a new game. The diagram includes the interactions between the different layers.

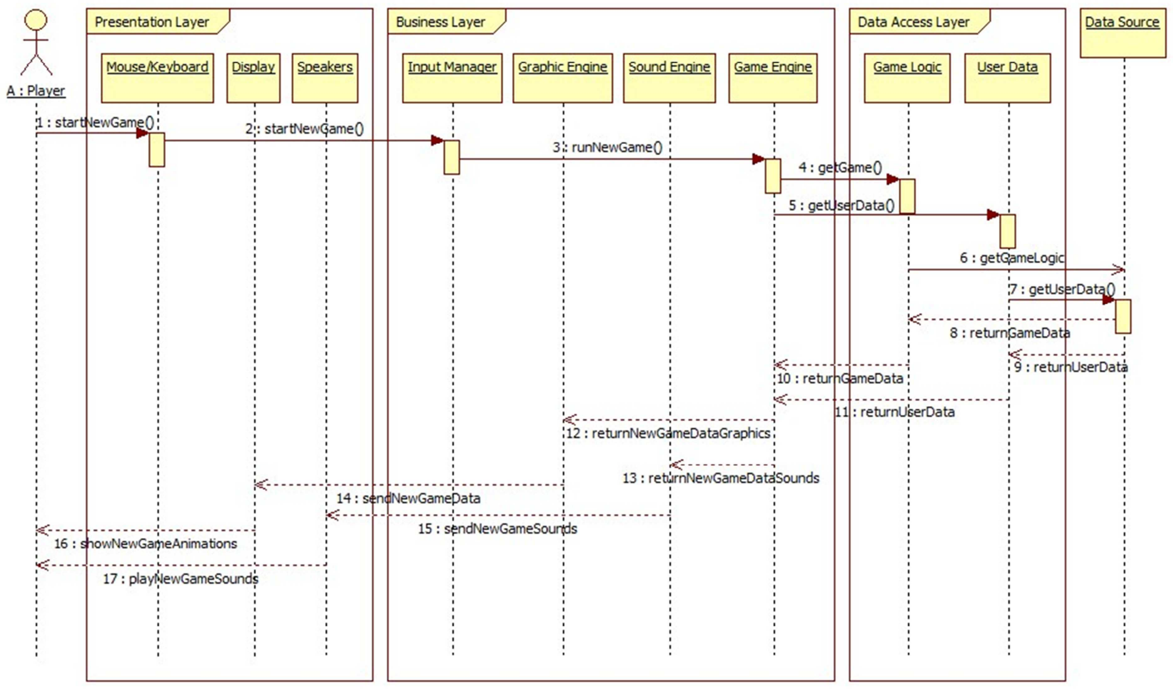


Figure 6-2: UML Sequence diagram – start a new game

In this layer we apply the “strategy design pattern” [21] in order to manage each type of activity. This pattern allows us to achieve a maintainable, scalable and high performance application. Additionally, an artificial intelligence algorithm called A* (A Star) [16] will be used in order to find the shortest path between two geographical locations in the factory. For instance, when a transport activity occurs, A* finds the path between the initial and final stations to move the requested part.

The business layer will create as many threads as necessary to allow the presentation layer to show a concurrent animation of being picked up and movements at parts around the plant. Also, the business layer contains a discrete event engine called SimPack, which is used to manage the events in the game. For example, (1) we use a periodic event managed by SimPack to validate and execute all the available activities that are obtained from the database; (2) when the game engine executes an activity, it schedules a future event in SimPack queue [15], and it must finish in some specific time to release any resources locked and make them available to other events.

6.1.4. Data Access Layer

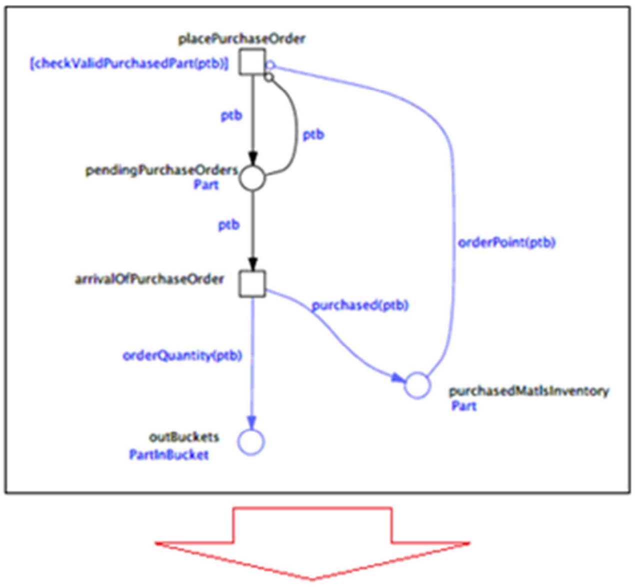
This bottom layer serves as the middleware between the application and the database. Whenever the application wants to run a new game or find out something about a user, it calls this layer to access the database and retrieve the required information. The layer applies the “façade design pattern,” [21] in order to simplify the complexity when other classes try to obtain all the information stored in the database. The “singleton pattern” is used when a class requires a large amount of memory and network to manage the information of the database.

The game database will be completely designed and built in the MySQL database engine. It will contain the game actors for each level, as well as the user data. When the user completes a new phase, his or her profile is updated, so that the next time the game runs it will start at the same point that the user left off.

6.2. Game Activities based on Petri Nets Models

Given all the different types of activities that can be executed in a real factory, we have summarized them into 4 different activities in the game. These activity types are: purchase, transport, operation, and shipping. These activities allow us to execute a complete production cycle beginning with the purchasing and storage of raw materials, assembling and storage the new product, and finally shipping the finished product.

At a conceptual level we have used *Petri Net* [28] to model these activities that will be implemented in the game. Once we understood the operation of the activities, we created a specific state machine for each activity which was then translated into Java code to be incorporated in the game. Each step in the state machine represents a state in the Petri Net model. For example, as shown in Figure 6-3, a purchase will be completed if and only if the validation was successful.



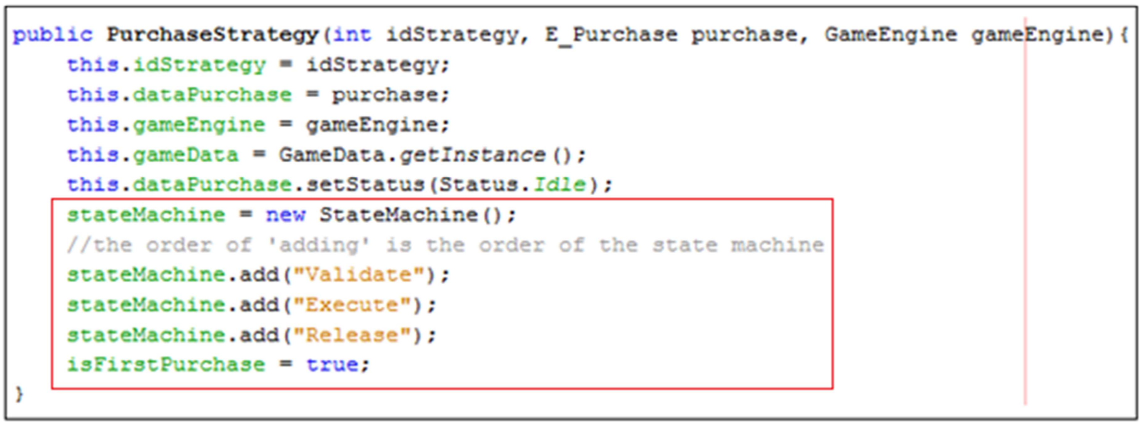
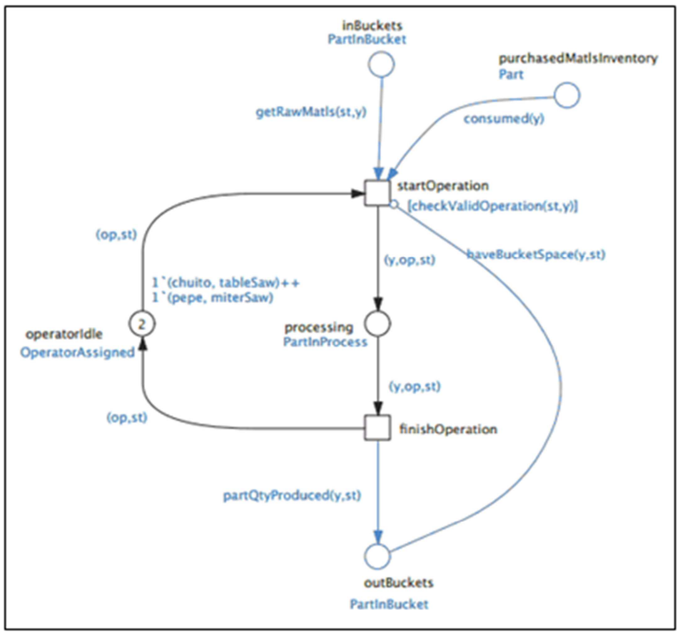
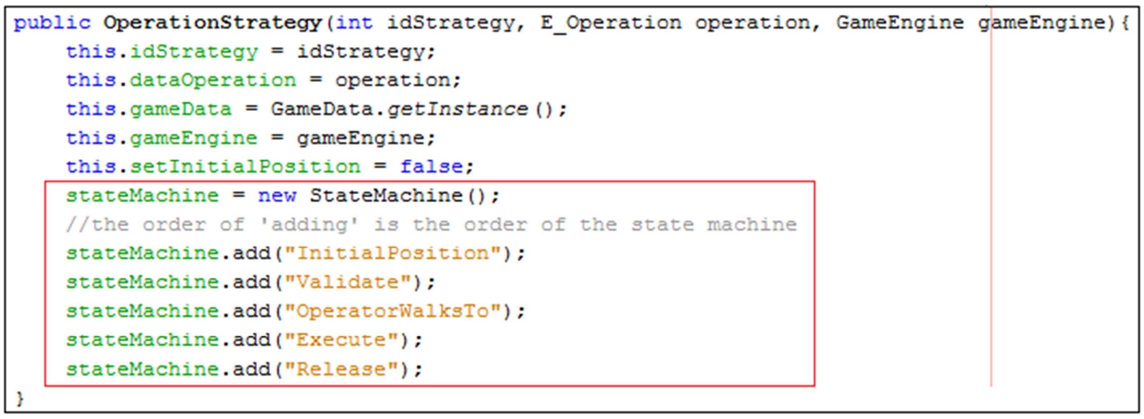

```
public PurchaseStrategy(int idStrategy, E_Purchase purchase, GameEngine gameEngine){
    this.idStrategy = idStrategy;
    this.dataPurchase = purchase;
    this.gameEngine = gameEngine;
    this.gameData = GameData.getInstance();
    this.dataPurchase.setStatus(Status.Idle);
    stateMachine = new StateMachine();
    //the order of 'adding' is the order of the state machine
    stateMachine.add("Validate");
    stateMachine.add("Execute");
    stateMachine.add("Release");
    isFirstPurchase = true;
}
```

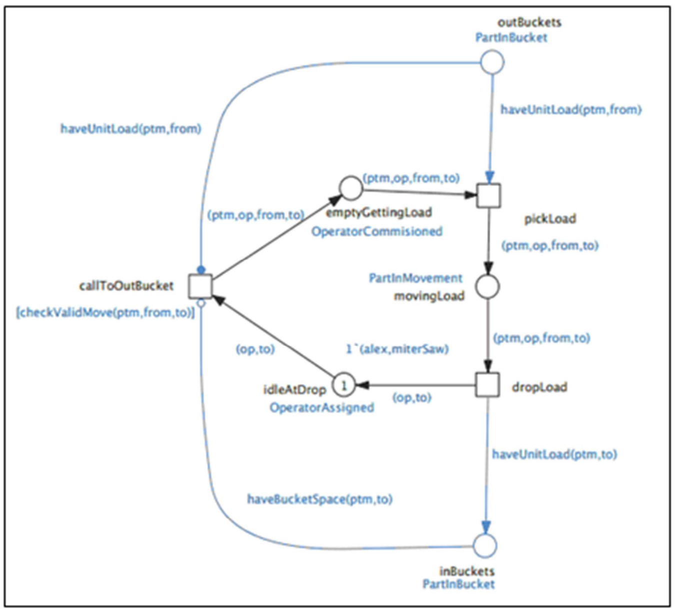
Figure 6-3: Purchase activity: Petri Net (top) and Java state machine (bottom)





```
public OperationStrategy(int idStrategy, E_Operation operation, GameEngine gameEngine){
    this.idStrategy = idStrategy;
    this.dataOperation = operation;
    this.gameData = GameData.getInstance();
    this.gameEngine = gameEngine;
    this.setInitialPosition = false;
    stateMachine = new StateMachine();
    //the order of 'adding' is the order of the state machine
    stateMachine.add("InitialPosition");
    stateMachine.add("Validate");
    stateMachine.add("OperatorWalksTo");
    stateMachine.add("Execute");
    stateMachine.add("Release");
}
```

Figure 6-4: Operation activity: Petri Net (top) and Java state machine (bottom)



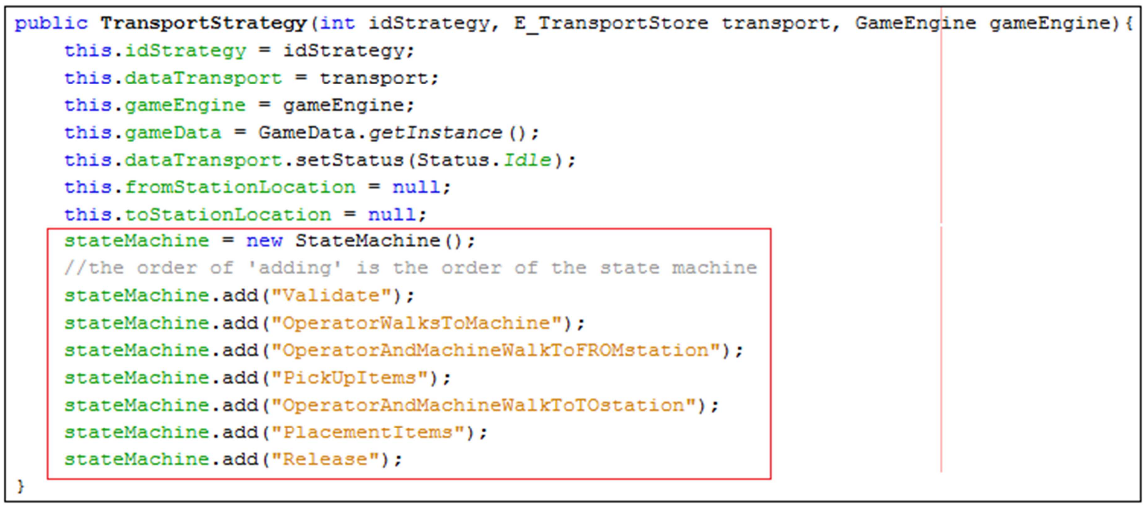

```java
public TransportStrategy(int idStrategy, E_TransportStore transport, GameEngine gameEngine){
    this.idStrategy = idStrategy;
    this.dataTransport = transport;
    this.gameEngine = gameEngine;
    this.gameData = GameData.getInstance();
    this.dataTransport.setStatus(Status.Idle);
    this.fromStationLocation = null;
    this.toStationLocation = null;
    stateMachine = new StateMachine();
    //the order of 'adding' is the order of the state machine
    stateMachine.add("Validate");
    stateMachine.add("OperatorWalksToMachine");
    stateMachine.add("OperatorAndMachineWalkToFROMstation");
    stateMachine.add("PickUpItems");
    stateMachine.add("OperatorAndMachineWalkToTOstation");
    stateMachine.add("PlacementItems");
    stateMachine.add("Release");
}
```

Figure 6-5: Transport activity: Petri Net (left) and Java state machine (right)

On the other hand, each game level has many different activities that have been loaded from the database. Therefore, in order to manage them as efficiently as possible, we developed the process of activity execution shown in Figure 6-6, which is carried out by the game engine, the discrete event machine plus some additional classes.

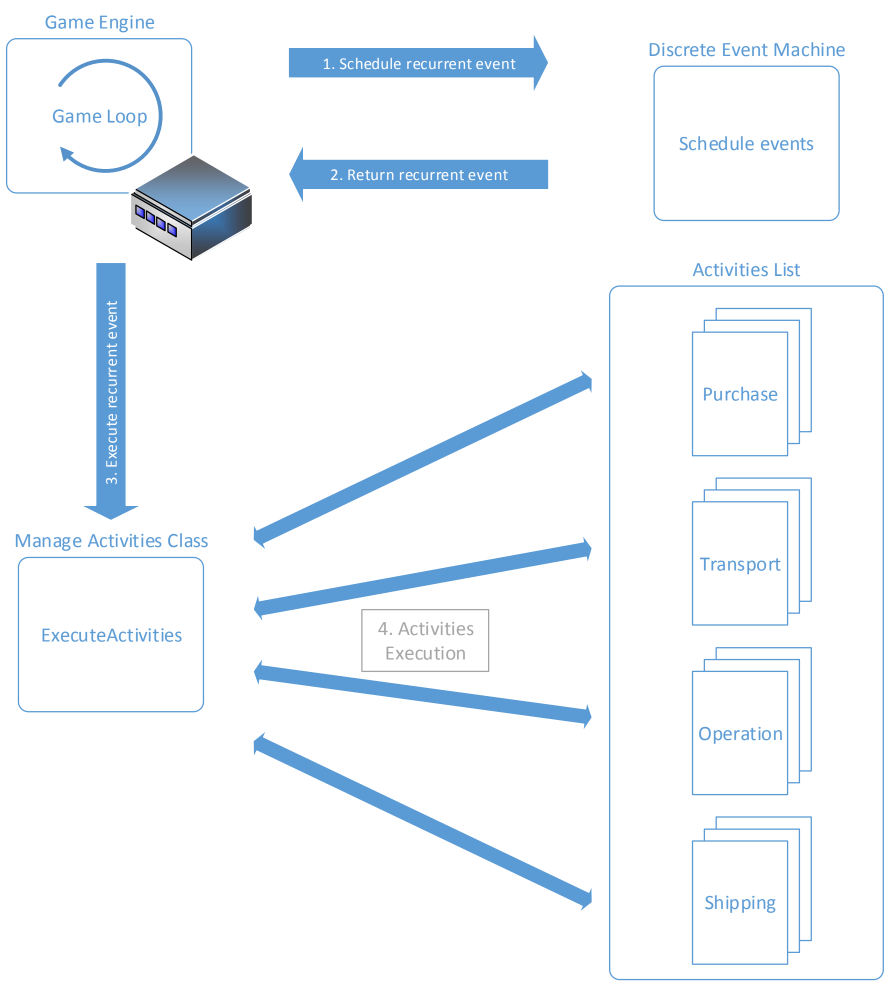


Figure 6-6: Management of activities in Java

According to Figure 4 above, this process takes 4 steps to execute activities. First, the game engine schedules a recurrent event in the discrete event machine. Second, once this event finishes, the discrete event machine returns it to the game engine. Third, the engine calls the method *ExecuteActivities()* in the *ManageActivities* class. Fourth, this method tries to execute the current step in the state machine of each activity. For example: *validate* requirements of a transport activity, or *release* resources of an operation activity. The same activity can be executed many times simultaneously and independently, depending of the availability of resources that it requires.

6.3. Package Diagram and Design Patterns

This section shows the package diagram of the project, including the dependencies among packages and the main classes in each package. It also describes the three design patterns which have been used in the development of the game.

6.3.1. Package Diagram

Figure 6-3 shows the relation between the principal packages in the game. These packages have been organized as efficiently as possible by grouping together classes with similar functionalities. Most packages have been developed by the author, except for SimPack and A* Pathfinding, which have been adapted from the public domain with a few customizations required for the game.

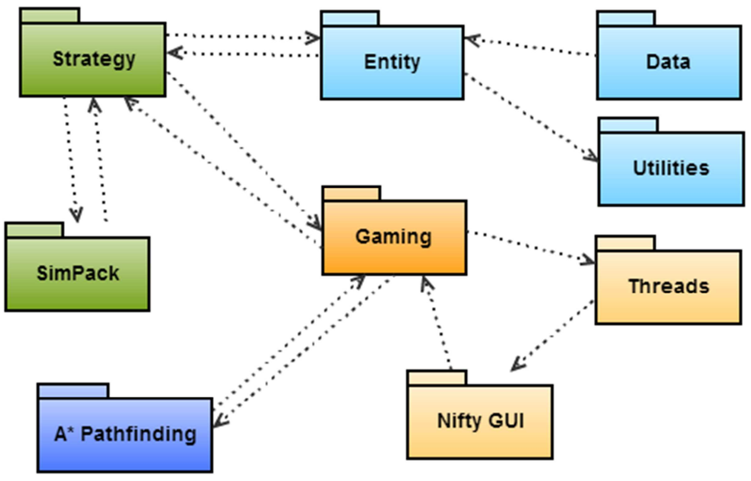


Figure 6-7: Overall game – Package diagram

Various design patterns have been applied in the game, in order to reduce the use of excessive memory, which will be explained in upcoming sections. However, the *Threads* package is used and instanced every time that a class is required, because these classes control each independent graphic animation within a different thread.

The *A* Pathfinding* package is used each time the game engine needs a new path between two different objects (e.g. an operator and machine, machine and station, an operator and station, etc.) The *SimPack* package is instanced only once because it has to manage all activities in process. The *Strategy* package manages all the different ongoing activity in the current game level (e.g. purchase activities, transports manufacturing, operations and shipments.) Finally, *Nifty GUI* package manages every screen and GUI control available to the user. Table 6-1 shows the classes used from each package.

Table 6-1: Package diagram detailed

Package	**Classes**		
Gaming	GameEngine OpeMaMovingTo	GameData OperatorWalksTo	InputManager TerrainMap
Threads	CloseGame MachineAnimation	GameDataLoading StationAnimation	UserDataLoading UpdateSlotsStorage
Nifty GUI	InitialMenu NewUserScreen	ForgotYourPassword MainMenu	NewGameMenu OptionsMenu
Nifty GUI Controls	ActivityControl FlowChartControl MachineControl OverallControl StationControl SupplierControl	AssignOperator GameLogControl OperatorControl PartControl StorageCostControl UnitLoadControl	CharactersControl GameSetupControl OrderControl PriorityControl StorageStationControl
A* Pathfinding	AStarHeuristic PathFinder	AStarPathFinder TileMap	Path ClosestHeuristic

SimPack	FutureEventList	Token	SimEvent
Strategy	PurchaseStrategy	ManageEvents	OperationStrategy
	TransportStrategy	ShipStrategy	StateMachine
Data	D_Activity	D_AssenblyDetails	D_Bucket
	D_Catalog	D_Game	D_Machine
	D_Operation	D_Operator	D_Order
	D_Part	D_Player	D_Purchase
	D_Ship	D_Skill	D_Station
	D_Supplier	D_Terrain	D_TerrainReserved
	D_ToUpdate	D_Transport	MySqlConn
	SQLiteConn	SQLiteUtilities	
Entity	E_Activity	E_AssemblyDetails	E_Bucket
	E_Catalog	E_Event	E_Game
	E_Machine	E_Operation	E_Operator
	E_Order	E_Part	E_Player
	E_PlayerLog	E_Purchase	E_Ship
	E_Skill	E_Slot	E_Station
	E_Supplier	E_Terrain	E_TerrainReserved
	E_ToUpdate	E_Transport	
Utilities	Actions	Direction	Distributions
	GameCategory	GameSounds	GameStatus
	GameTables	GameType	MessageType
	Messages	ObjectState	OperatorCategory
	OrderStates	Pair	Params
	PasswordGenerator	SendEmail	SlotStatus
	Sounds	StationType	Status
	TypeActivity	TypeElements	Utils

6.3.2. Strategy Pattern

The strategy design pattern is used in the game to control every activity during each game level. The game manages four types of activities: purchase, operation, transport and shipping. These are represented by corresponding classes, one type of activity for each class (Figure 6-4). However, since the game engine selects an activity at runtime, these activities should be encapsulated to make them interchangeable. The class called *ManageEvents* controls a list of such activities by means at an interface called *EventStrategy*.

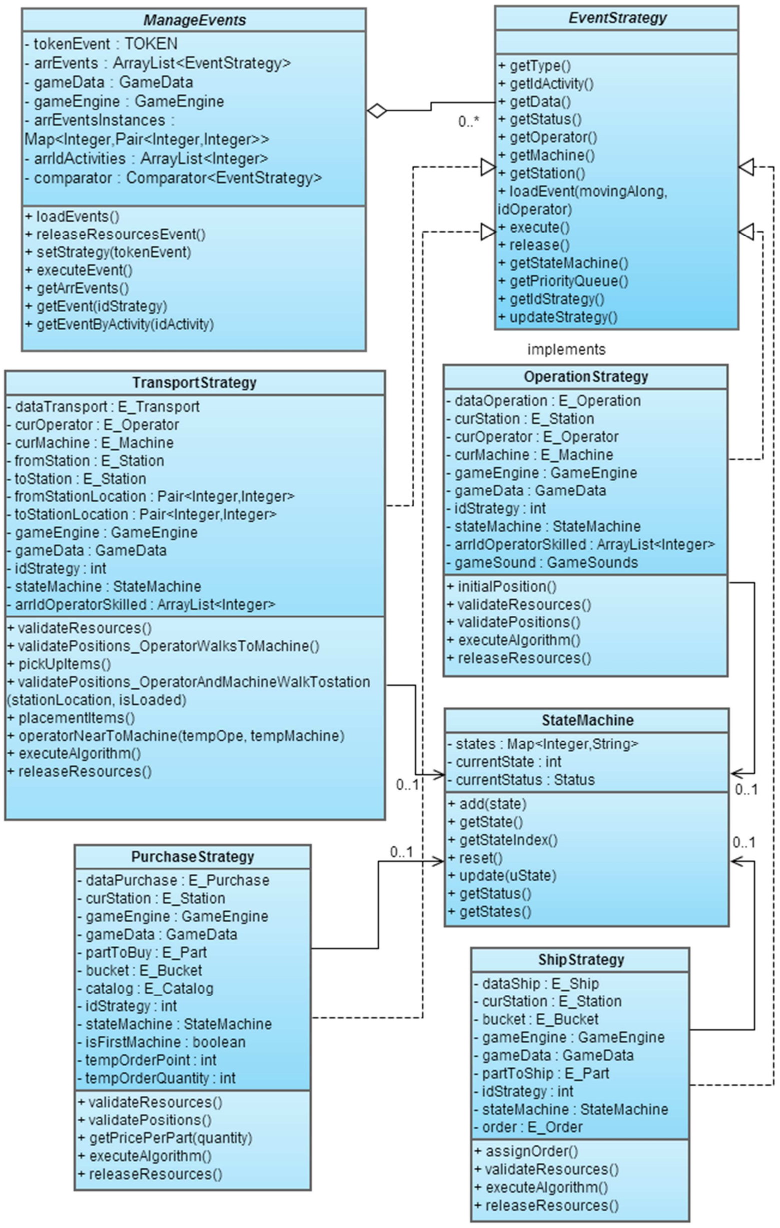


Figure 6-8: Strategy pattern – Class diagram

The class called *StateMachine*, which is related to every activity class, provides a configurable state machine for each activity, since each activity invokes a unique sequence of steps to be completed. For instance: a purchase activity has three (3) states, i.e. validate, execute and release. On the other hand, a transport activity has seven (7) states, i.e. validate, operator walks to machine, operator and machine walk to initial station, operator loads items into machine, operator and machine walk to final station, operator unloads machine, and operator release machine.

6.3.3. Façade Pattern

A design pattern could be composed of many classes yet, what matters is these classes are structured and how they work together to solve a problem. In a complex system (i.e. many different classes with many functions), which requires performing a function, it will need to call each class required. In order to avoid this, we use the façade design pattern, which provides a simplified interface to a complex system.

In the game development, this pattern is applied in the data management lager, which set its information from a relational database. This complex system is comprised of 38 classes located in two packages: *entity* and *data*. The *GameData* class provides a simple interface to this information (Figure 6-5.) This class manages all references and variables for each of the 38 classes, and provides some public methods (e.g. loadGamesByType(), createGame(), manageGame(), etc.)

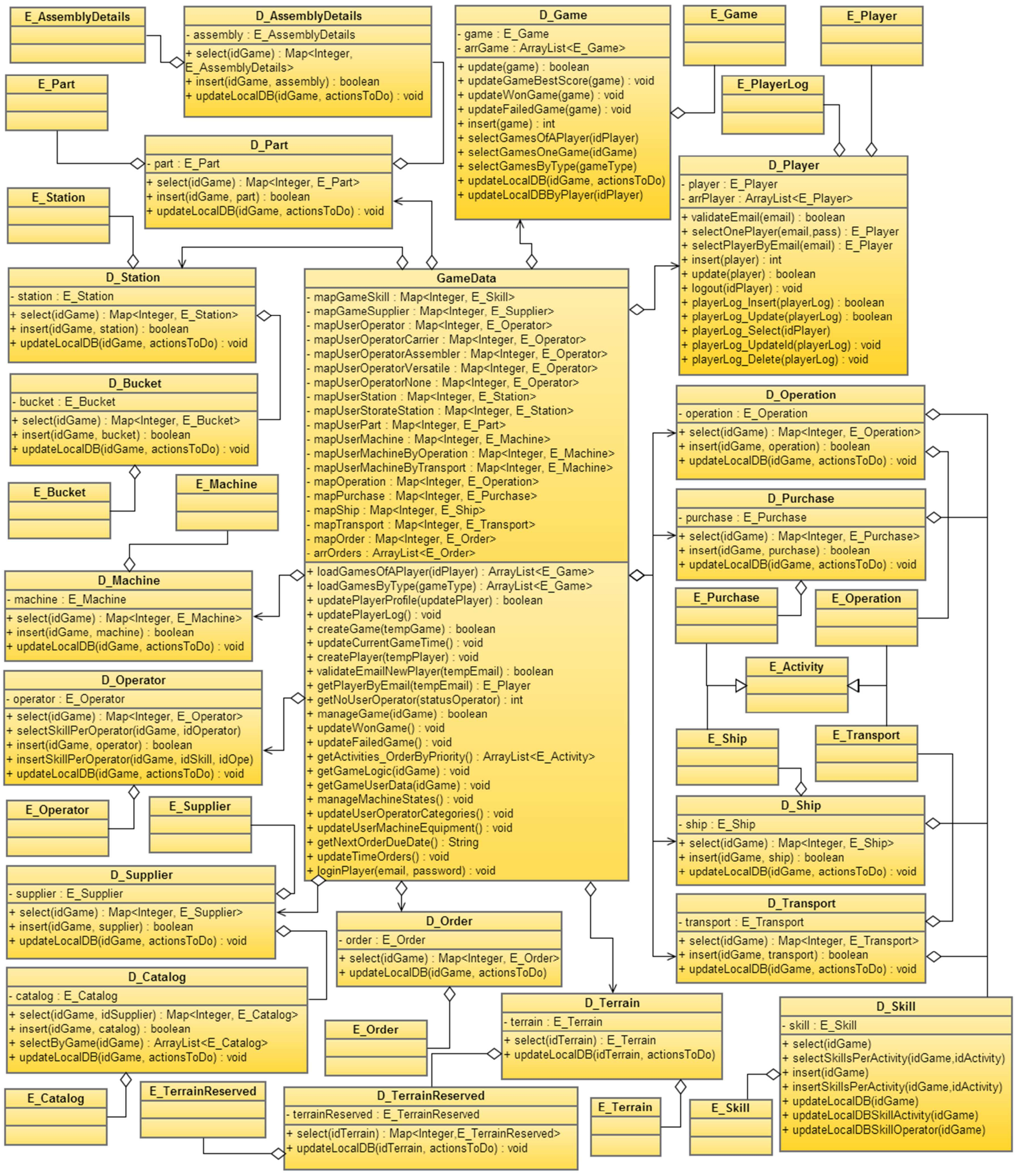

E_AssemblyDetails
D_AssemblyDetails
- assembly : E_AssemblyDetails
+ select(idGame) : Map<Integer, E_AssemblyDetails>
+ insert(idGame, assembly) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
D_Game
- game : E_Game
- arrGame : ArrayList<E_Game>
+ update(game) : boolean
+ updateGameBestScore(game) : void
+ updateWonGame(game) : void
+ updateFailedGame(game) : void
+ insert(game) : int
+ selectGamesOfAPlayer(idPlayer)
+ selectGamesOneGame(idGame)
+ selectGamesByType(gameType)
+ updateLocalDB(idGame, actionsToDo)
+ updateLocalDBByPlayer(idPlayer)
E_Game
E_Player
E_PlayerLog
E_Part
D_Part
- part : E_Part
+ select(idGame) : Map<Integer, E_Part>
+ insert(idGame, part) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
E_Station
D_Player
- player : E_Player
- arrPlayer : ArrayList<E_Player>
+ validateEmail(email) : boolean
+ selectOnePlayer(email,pass) : E_Player
+ selectPlayerByEmail(email) : E_Player
+ insert(player) : int
+ update(player) : boolean
+ logout(idPlayer) : void
+ playerLog_Insert(playerLog) : boolean
+ playerLog_Update(playerLog) : boolean
+ playerLog_Select(idPlayer)
+ playerLog_UpdateId(playerLog) : void
+ playerLog_Delete(playerLog) : void
D_Station
- station : E_Station
+ select(idGame) : Map<Integer, E_Station>
+ insert(idGame, station) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
GameData
- mapGameSkill : Map<Integer, E_Skill>
- mapGameSupplier : Map<Integer, E_Supplier>
- mapUserOperator : Map<Integer, E_Operator>
- mapUserOperatorCarrier : Map<Integer, E_Operator>
- mapUserOperatorAssembler : Map<Integer, E_Operator>
- mapUserOperatorVersatile : Map<Integer, E_Operator>
- mapUserOperatorNone : Map<Integer, E_Operator>
- mapUserStation : Map<Integer, E_Station>
- mapUserStorateStation : Map<Integer, E_Station>
- mapUserPart : Map<Integer, E_Part>
- mapUserMachine : Map<Integer, E_Machine>
- mapUserMachineByOperation : Map<Integer, E_Machine>
- mapUserMachineByTransport : Map<Integer, E_Machine>
- mapOperation : Map<Integer, E_Operation>
- mapPurchase : Map<Integer, E_Purchase>
- mapShip : Map<Integer, E_Ship>
- mapTransport : Map<Integer, E_Transport>
- mapOrder : Map<Integer, E_Order>
- arrOrders : ArrayList<E_Order>
+ loadGamesOfAPlayer(idPlayer) : ArrayList<E_Game>
+ loadGamesByType(gameType) : ArrayList<E_Game>
+ updatePlayerProfile(updatePlayer) : boolean
+ updatePlayerLog() : void
+ createGame(tempGame) : boolean
+ updateCurrentGameTime() : void
+ createPlayer(tempPlayer) : void
+ validateEmailNewPlayer(tempEmail) : boolean
+ getPlayerByEmail(tempEmail) : E_Player
+ getNoUserOperator(statusOperator) : int
+ manageGame(idGame) : boolean
+ updateWonGame() : void
+ updateFailedGame() : void
+ getActivities_OrderByPriority() : ArrayList<E_Activity>
+ getGameLogic(idGame) : void
+ getGameUserData(idGame) : void
+ manageMachineStates() : void
+ updateUserOperatorCategories() : void
+ updateUserMachineEquipment() : void
+ getNextOrderDueDate() : String
+ updateTimeOrders() : void
+ loginPlayer(email, password) : void
D_Bucket
- bucket : E_Bucket
+ select(idGame) : Map<Integer, E_Bucket>
+ insert(idGame, bucket) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
D_Operation
- operation : E_Operation
+ select(idGame) : Map<Integer, E_Operation>
+ insert(idGame, operation) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
E_Bucket
E_Machine
D_Purchase
- purchase : E_Purchase
+ select(idGame) : Map<Integer, E_Purchase>
+ insert(idGame, purchase) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
D_Machine
- machine : E_Machine
+ select(idGame) : Map<Integer, E_Machine>
+ insert(idGame, machine) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
E_Purchase
E_Operation
E_Activity
D_Operator
- operator : E_Operator
+ select(idGame) : Map<Integer, E_Operator>
+ selectSkillPerOperator(idGame, idOperator)
+ insert(idGame, operator) : boolean
+ insertSkillPerOperator(idGame, idSkill, idOpe)
+ updateLocalDB(idGame, actionsToDo) : void
E_Ship
E_Transport
E_Operator
E_Supplier
D_Ship
- ship : E_Ship
+ select(idGame) : Map<Integer, E_Ship>
+ insert(idGame, ship) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
D_Supplier
- supplier : E_Supplier
+ select(idGame) : Map<Integer, E_Supplier>
+ insert(idGame, supplier) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
D_Order
- order : E_Order
+ select(idGame) : Map<Integer, E_Order>
+ updateLocalDB(idGame, actionsToDo)
D_Transport
- transport : E_Transport
+ select(idGame) : Map<Integer, E_Transport>
+ insert(idGame, transport) : boolean
+ updateLocalDB(idGame, actionsToDo) : void
D_Catalog
- catalog : E_Catalog
+ select(idGame, idSupplier) : Map<Integer, E_Catalog>
+ insert(idGame, catalog) : boolean
+ selectByGame(idGame) : ArrayList<E_Catalog>
+ updateLocalDB(idGame, actionsToDo) : void
D_Terrain
- terrain : E_Terrain
+ select(idTerrain) : E_Terrain
+ updateLocalDB(idTerrain, actionsToDo)
D_Skill
- skill : E_Skill
+ select(idGame)
+ selectSkillsPerActivity(idGame,idActivity)
+ insert(idGame)
+ insertSkillsPerActivity(idGame,idActivity)
+ updateLocalDB(idGame)
+ updateLocalDBSkillActivity(idGame)
+ updateLocalDBSkillOperator(idGame)
E_Order
E_Catalog
E_TerrainReserved
D_TerrainReserved
- terrainReserved : E_TerrainReserved
+ select(idTerrain) : Map<Integer,E_TerrainReserved>
+ updateLocalDB(idTerrain, actionsToDo) : void
E_Terrain
E_Skill

Figure 6-9: Façade pattern – Class diagram

6.3.4. Singleton Pattern

This design pattern is usually applied to avoid unnecessarily creating many instances of a class. All other classes use a single instance of the class every time. The pattern coordinates when different threads try to obtain information from this instance at the same time, by making the main method "synchronized". The *GameData* class holds a lot of information and thus requires a lot of memory; so applying singleton pattern avoids duplicating this data. Figure 6-6 below shows the relation between the *GameData* class and the *GameEngine* class indicateing that it uses the singleton pattern.

The *MySqlConn* and *SQLiteConn* classes also use the singleton pattern, and are called each time the game engine needs to communicate with either database.

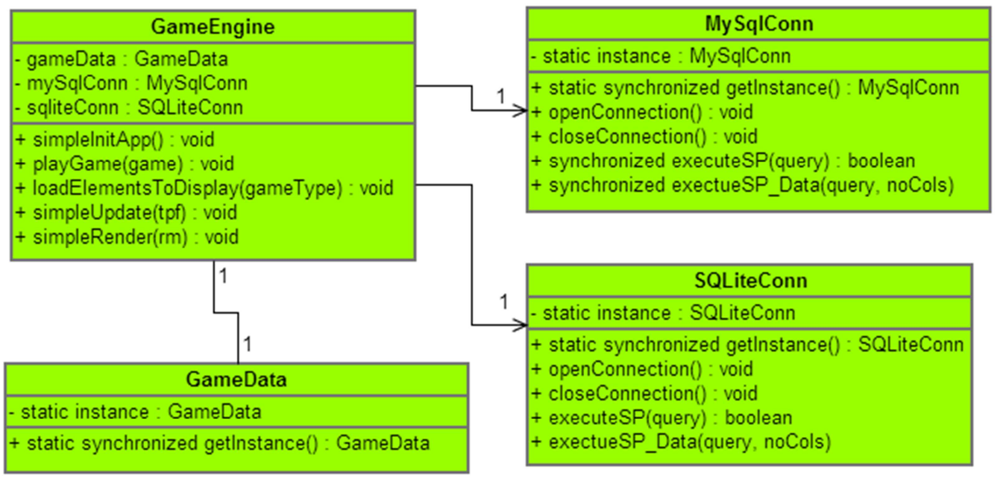

GameEngine
- gameData : GameData
- mySqlConn : MySqlConn
- sqliteConn : SQLiteConn
+ simpleInitApp() : void
+ playGame(game) : void
+ loadElementsToDisplay(gameType) : void
+ simpleUpdate(tpf) : void
+ simpleRender(rm) : void
MySqlConn
- static instance : MySqlConn
+ static synchronized getInstance() : MySqlConn
+ openConnection() : void
+ closeConnection() : void
+ synchronized executeSP(query) : boolean
+ synchronized exectueSP_Data(query, noCols)
SQLiteConn
- static instance : SQLiteConn
+ static synchronized getInstance() : SQLiteConn
+ openConnection() : void
+ closeConnection() : void
+ executeSP(query) : boolean
+ exectueSP_Data(query, noCols)
GameData
- static instance : GameData
+ static synchronized getInstance() : GameData
1
1
1
1

Figure 6-10: Singleton pattern – Class diagram

6.4. Database Design

The database design changed continuously during game development. Initially, the game only managed the MySQL engine as the remote database, storing all game and user data. When a local user started a new game level the engine would have to download all required data from the remote database. This process took substantial time, depending of the available Internet bandwidth. As a result, SQLite was used as the local database cache while MySQL remained as the remote database holding the main data.

On the other hand, the database design allowed us to add and update each game level from the database. This means that we did not need to modify nor upgrade anything in the game source code to incorporate new levels into the game. From the beginning of the game development, the database has always maintained the game logic and user data. Therefore, once the player starts a new game level, the game engine dynamically creates the virtual reality in accordance to the information stored in the database. However, this advantage is not so easy to perform, given that it has been designed under a relational database model. So if we want to add or modify a game level, we must validate different constraints among tables, primary and foreign keys of every record, and different table hierarchies.

6.4.1. MySQL and SQLite Design

6.4.1.1. Principal Tables

Figure 6-7 displays the Entity Relationship Model (ERM) diagram for the game application.

The ERM diagram depicts core tables, which are required for game management, to maintain each game level with their dependencies, and to store each user data with their statistical data usage. Relationships between tables are meant to create non-duplicate users, to store the game level data independently of each user, to save the usage data for each user, while maintaining both local and remote databases synchronized.

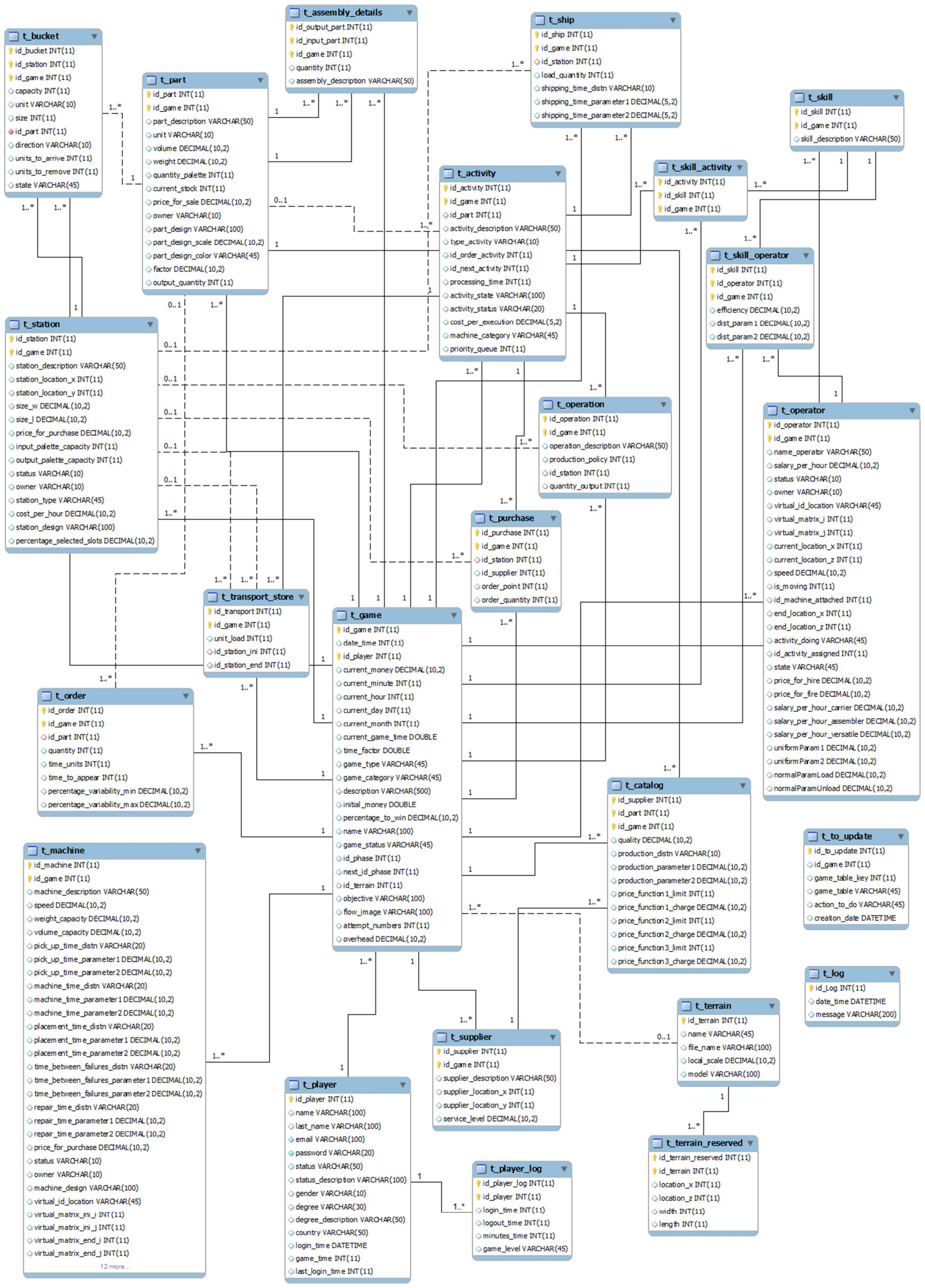


Figure 6-11: Entity relationship diagram

6.4.1.2. Principal Routines

The remote database (managed by MySQL) is storing numerous routines that allow the game engine to be managed satisfactorily and efficiently. These routines are built to maintain the data for each table, and for that reason each table at least 2 basic routines (i.e. select routine and update routine). The rest of the routines represent more complex queries.

One drawback of the SQLite engine is that it does not store routines. In that case, queries are stored inside the source code. If for some reason the table changes its structure, then routines located in the source code will be updated and implying in a new game release.

6.4.2. Databases Synchronization Process

This synchronization process is the main tool that allows achieving three main tasks: (1) to gather usage data for each user into the remote database, (2) to update the game data for each user machine from the remote database, and (3) to update current game levels or to add new game levels from remote database to every local database. Figure 6-8 below illustrates the communication between the database server and different user machines (i.e. desktop and laptop computers), where each one is executing the sync process. More information about this process can be found in Section 5.3.

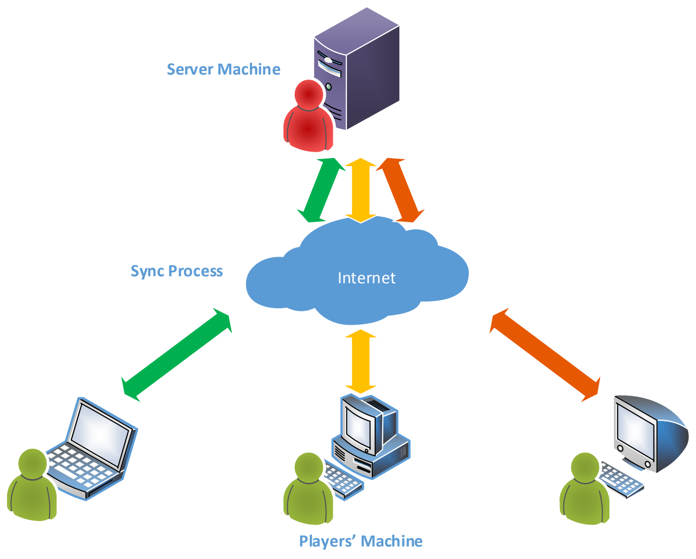


Figure 6-12: Synchronization process between databases

6.5. Detailed Description of Components

This section will provide additional information about important software components that have been used in the development of the game: included as Java threads, A* algorithm, SimPack, and Nifty GUI.

6.5.1. Java Threads

Java SE provides different methods to implement threads in a program. The project is using classes that are subclasses of the *Thread* class. Also the game contains a package designated exclusively for *Thread* subclasses (Table 6-1.) Threads in the project are in charge of multiple independent graphics animations, given that each animation should not disrupt the normal execution of the other animations. Also, these threads execute different functionalities concurrently such as: triggering different sync processes, updating the use of slots for storage, and controlling the inactivity time of the user during the game.

6.5.2. A* Algorithm

The A* algorithm is used for calculate routes in the game environment. This is a set of classes which control internally a bi-dimensional map with coordinates for each object, indicating whether coordinates are available, blocked, or step. The algorithm finds the shortest path possible between two given coordinates. The repeated use of this algorithm may consume many resources, i.e. RAM memory and CPU. Therefore, the game engine only calls the algorithm when it is absolutely necessary. Figure 6-9 shows five (5) different paths found by A* among different stations.

A* is called when an instance of some transport activity is ready to be executed. It means, once the operator is done loading parts on the equipment, the next step is to move the equipment to the final station. However, before the equipment is moved, this activity executes the algorithm and finds a path. Finally, the material handler with the equipment starts moving along the path computed by A*.

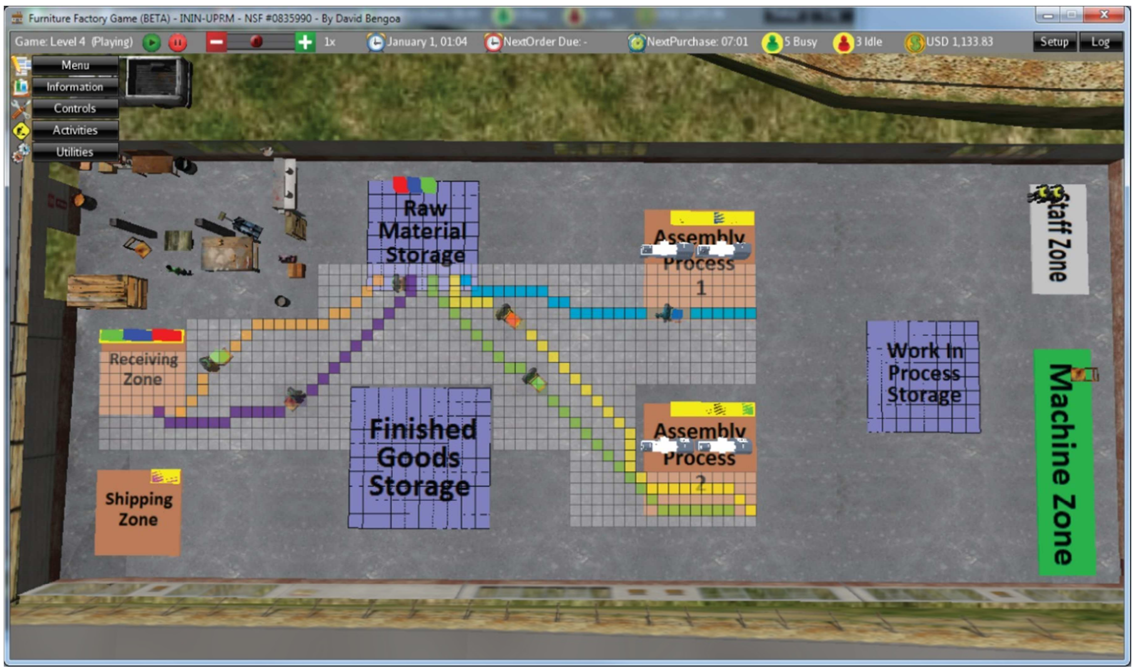


Figure 6-13: A* pathfinding example in the game environment

6.5.3. SimPack

The SimPack library supplies the game engine with an easy mechanism for scheduling multiple concurrent activities. This library implements a list the scheduled activities, and, in case it is required, the automatic update of the due date of each activity (e.g. if the game speed goes faster or slower.)

The main class in the game has on override method called “simpleUpdate”, this method acts as an infinite loop because the game engine calls it several times per second. For gaming purposes, inside this method the engine verifies if some activity in SimPack has completed its due date. If this is the case, this activity is removed of the future event list of SimPack and carried out in the game. SimPack manages two (2) types of activities, *recurrent activities* and *different activities* as shown in Figure 6-10. Each one has a different meaning in the game.

A *recurrent activity* is scheduled each time unit (e.g. one second.) Once the activity is over two tasks are performed: (1) another similar activity is scheduled in the future event list of SimPack, and (2) the game engine executes a method in another class which has the function of validating and executing other game activities (e.g. purchase, transport, operation, and shipping.)

In accordance with *different activities*, once the game engine validates an activity successfully (e.g. purchase, transport, operation or shipping activity), then this activity has to perform another task which will take some time to be completed, e.g. three time units. Therefore, this activity is scheduled in the future event list of SimPack with a given due date. Finally, when this activity in the list is completed, it will execute a code according to the gaming activity, e.g. release resources, execute another task, update inventory of some parts, among others.

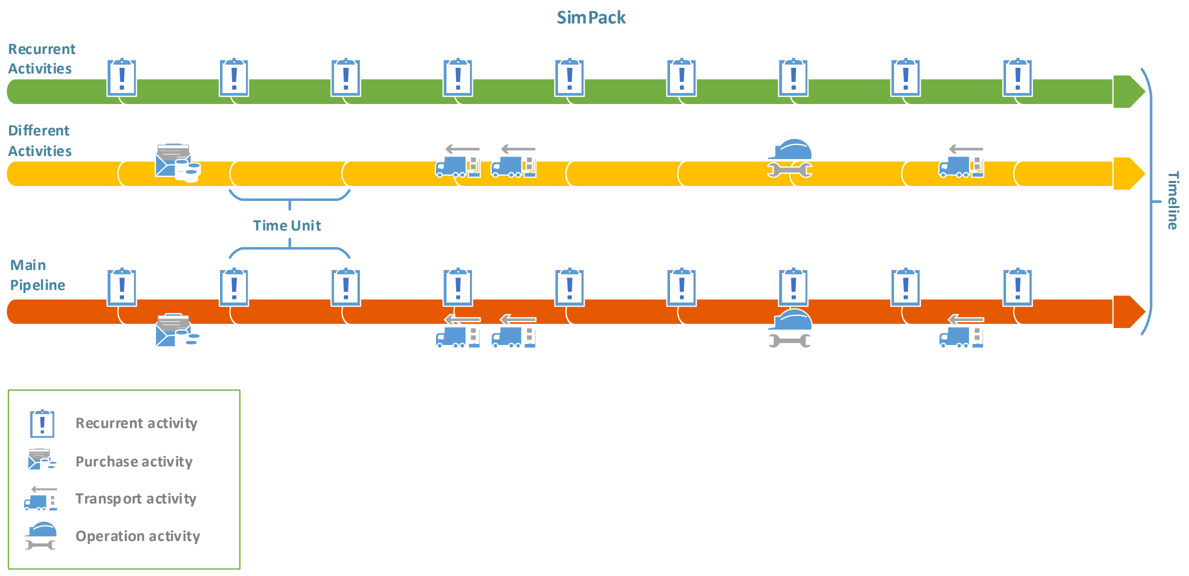
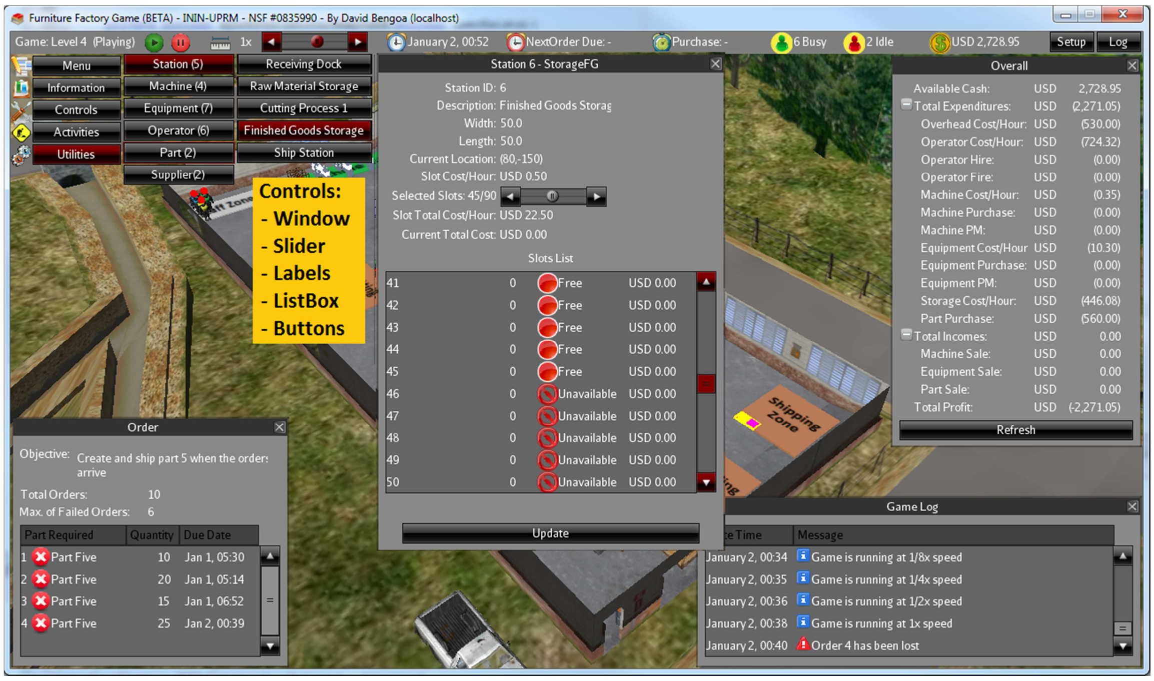


Figure 6-14: SimPack – discrete event machine in the game

6.5.4. Nifty GUI

This library provides the game with a way to create different panels with a variety at user controls. Figure 6-11 shows a subset at the controls available in Nifty GUI.

Nifty GUI allows the developer to create custom controls. It means by combining existing controls. For example, the standard listbox control only shows text, but it can be customized to show images, text, and buttons in a single row.



Station 6 - StorageFG

Station ID:	6
Description:	Finished Goods Storag
Width:	50.0
Length:	50.0
Current Location:	(80,-150)
Slot Cost/Hour:	USD 0.50
Selected Slots: 45/90	
Slot Total Cost/Hour:	USD 22.50
Current Total Cost:	USD 0.00

Slots List

41	0	Free	USD 0.00
42	0	Free	USD 0.00
43	0	Free	USD 0.00
44	0	Free	USD 0.00
45	0	Free	USD 0.00
46	0	Unavailable	USD 0.00
47	0	Unavailable	USD 0.00
48	0	Unavailable	USD 0.00
49	0	Unavailable	USD 0.00
50	0	Unavailable	USD 0.00

Overall

Available Cash:	USD	2,728.95
Total Expenditures:	USD	(2,271.05)
Overhead Cost/Hour:	USD	(530.00)
Operator Cost/Hour:	USD	(724.32)
Operator Hire:	USD	(0.00)
Operator Fire:	USD	(0.00)
Machine Cost/Hour:	USD	(0.35)
Machine Purchase:	USD	(0.00)
Machine PM:	USD	(0.00)
Equipment Cost/Hour	USD	(10.30)
Equipment Purchase:	USD	(0.00)
Equipment PM:	USD	(0.00)
Storage Cost/Hour:	USD	(446.08)
Part Purchase:	USD	(560.00)
Total Incomes:	USD	0.00
Machine Sale:	USD	0.00
Equipment Sale:	USD	0.00
Part Sale:	USD	0.00
Total Profit:	USD	(-2,271.05)

Order

Objective: Create and ship part 5 when the order: arrive

Total Orders: 10
Max. of Failed Orders: 6

Part Required	Quantity	Due Date
1 Part Five	10	Jan 1, 05:30
2 Part Five	20	Jan 1, 05:14
3 Part Five	15	Jan 1, 06:52
4 Part Five	25	Jan 2, 00:39

Game Log

e Time	Message
January 2, 00:34	Game is running at 1/8x speed
January 2, 00:35	Game is running at 1/4x speed
January 2, 00:36	Game is running at 1/2x speed
January 2, 00:38	Game is running at 1x speed
January 2, 00:40	Order 4 has been lost

Figure 6-15: GUI controls in the game environment

6.6. User Interface Design

The same user interface design is based on the software requirements presented in Section 5.3. This section describes the most notable user interfaces features.

6.6.1. Login and Main Menu

Figure 6-12 shows the login and main menu panel, the first screen that the player sees each time that he/she starts the game. The login process will require an email registered with a valid password. Once the user is authenticated, the game proceeds to synchronize the user data between the local and remote databases. The figure also shows the main menu consisting of several buttons that can be used to select among the options mentioned in Section 5.2.1.2.

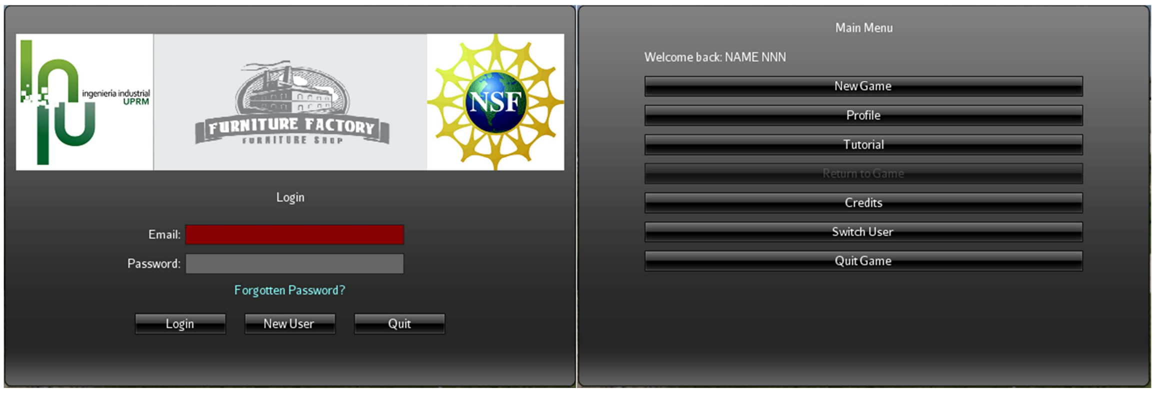
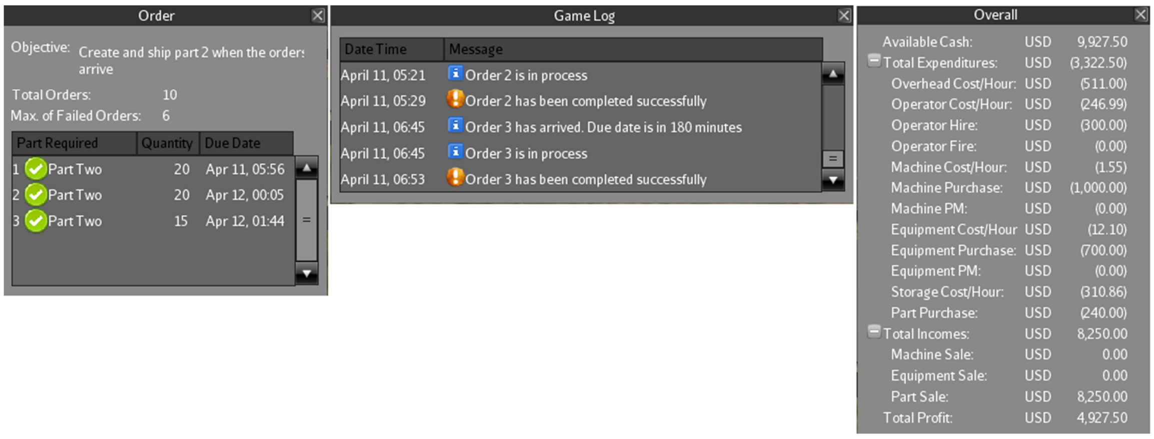


Figure 6-16: Login menu (left) and main menu (right)

6.6.2. Main Information Panels

Figure 6-13 is composed of three subpanels, which have different functionalities and allow the player to watch constantly the most critical information in the game. The Order subpanel (left) shows all pending and completed orders as they arrive during the game. Game log subpanel (center) allows the user to maintain each activity as they occur during the game (e.g. game is paused, you'll go bankrupt, order 1 has arrived, etc.) Overall subpanel (right) depicts detailed information about the costs in the game. Furthermore, this screen design avoids visual interference of the factory.



Order

Objective: Create and ship part 2 when the orders arrive

Total Orders: 10

Max. of Failed Orders: 6

	Part Required	Quantity	Due Date
1	✓ Part Two	20	Apr 11, 05:56
2	✓ Part Two	20	Apr 12, 00:05
3	✓ Part Two	15	Apr 12, 01:44

Game Log

Date Time	Message
April 11, 05:21	Order 2 is in process
April 11, 05:29	Order 2 has been completed successfully
April 11, 06:45	Order 3 has arrived. Due date is in 180 minutes
April 11, 06:45	Order 3 is in process
April 11, 06:53	Order 3 has been completed successfully

Overall

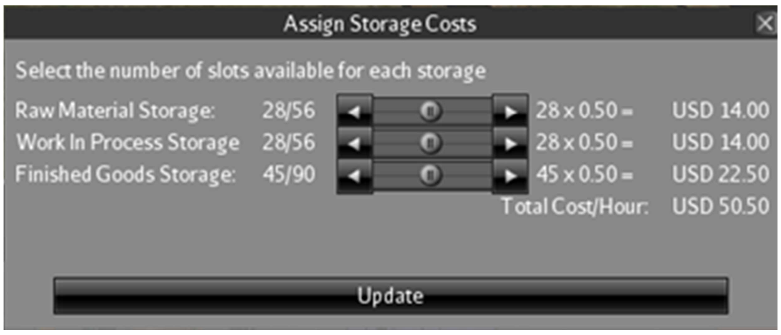
Available Cash:	USD	9,927.50
Total Expenditures:	USD	(3,322.50)
Overhead Cost/Hour:	USD	(511.00)
Operator Cost/Hour:	USD	(246.99)
Operator Hire:	USD	(300.00)
Operator Fire:	USD	(0.00)
Machine Cost/Hour:	USD	(1.55)
Machine Purchase:	USD	(1,000.00)
Machine PM:	USD	(0.00)
Equipment Cost/Hour	USD	(12.10)
Equipment Purchase:	USD	(700.00)
Equipment PM:	USD	(0.00)
Storage Cost/Hour:	USD	(310.86)
Part Purchase:	USD	(240.00)
Total Incomes:	USD	8,250.00
Machine Sale:	USD	0.00
Equipment Sale:	USD	0.00
Part Sale:	USD	8,250.00
Total Profit:	USD	4,927.50

Figure 6-17: Order, game log and overall screens merged in one screen

6.6.3. Control Screens

6.6.3.1. Allocate Storages

Figure 6-14 shows the screen that allows allocating an amount of slots available for storages. Selected slots will entail costs, which are calculated every hour. Also, an “update” button will commit changes in storage.



Assign Storage Costs

Select the number of slots available for each storage

Raw Material Storage:	28/56	28 x 0.50 =	USD 14.00
Work In Process Storage	28/56	28 x 0.50 =	USD 14.00
Finished Goods Storage:	45/90	45 x 0.50 =	USD 22.50
		Total Cost/Hour:	USD 50.50

Figure 6-18: Allocate storage screen

6.6.3.2. Assign Operators

The screen shown in Figure 6-15, allows the user to select an activity (left side list), then he or she will be able to choose material handlers, line operators, or versatile, depending of the role required by the activity. The value found between brackets, i.e. “(#)”, indicates the number of activities assigned and the number of operators assigned, for the operator and the activity, respectively.

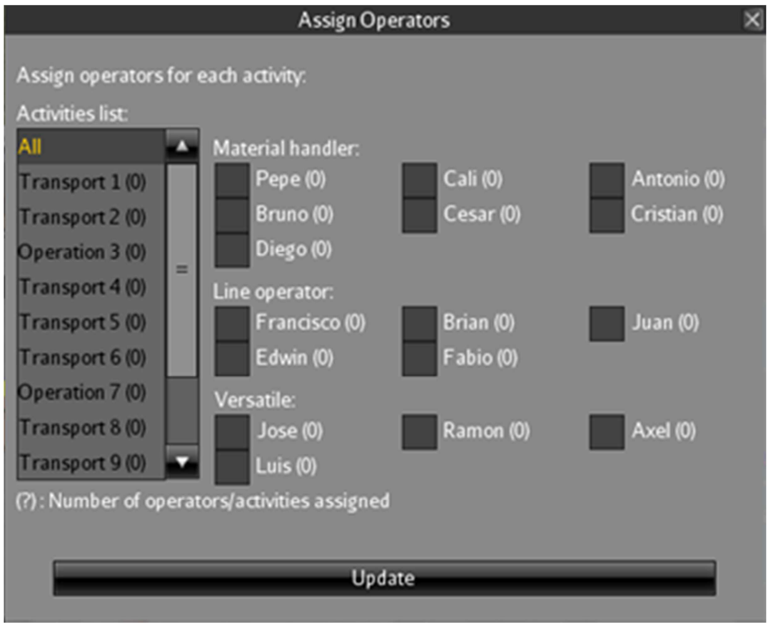
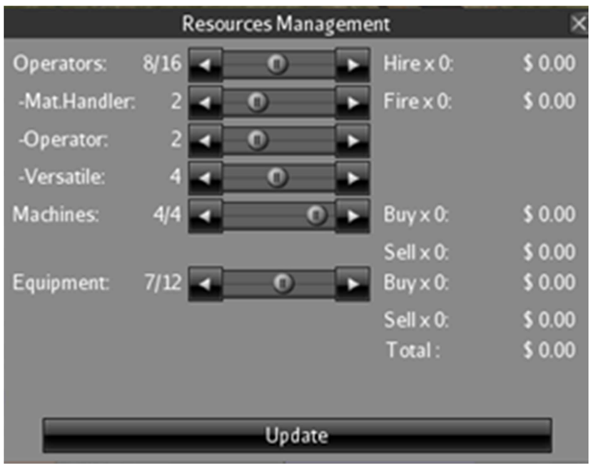


Figure 6-19: Assign workers screen

6.6.3.3. Resources

The resource management screen (Figure 6-16) is particularly crucial before starting the game. It allows users to hire and fire operators, and to buy and sell machines and equipment. Each decision of the user implies a cost and it will vary depending of the game level, for example: hiring two (2) operators, buying one (1) machine, and selling three (3) equipment. In addition, once the user hires operators, he or she has to choose if the operator is going to be a material handler an operator or versatile.

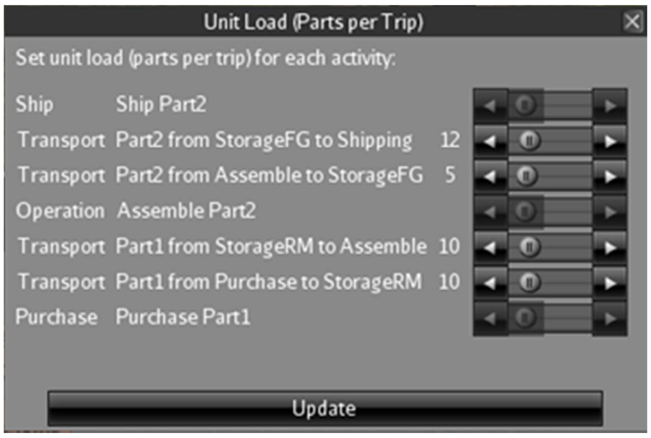

Resources Management

Operators:	8/16	Hire x 0:	$ 0.00
-Mat.Handler:	2	Fire x 0:	$ 0.00
-Operator:	2		
-Versatile:	4		
Machines:	4/4	Buy x 0:	$ 0.00
		Sell x 0:	$ 0.00
Equipment:	7/12	Buy x 0:	$ 0.00
		Sell x 0:	$ 0.00
		Total :	$ 0.00

Figure 6-20: Resources screen

6.6.3.4. Unit Load

The "unit load" screen is presented in Figure 6-17. The screen shows the list of activities for the current game level. The slider control is enabled only for transport activities, allowing the user to choose the bulk of parts to be moved in each activity. Also, the "update" button registers the changes in for each activity.


Unit Load (Parts per Trip)

Set unit load (parts per trip) for each activity:

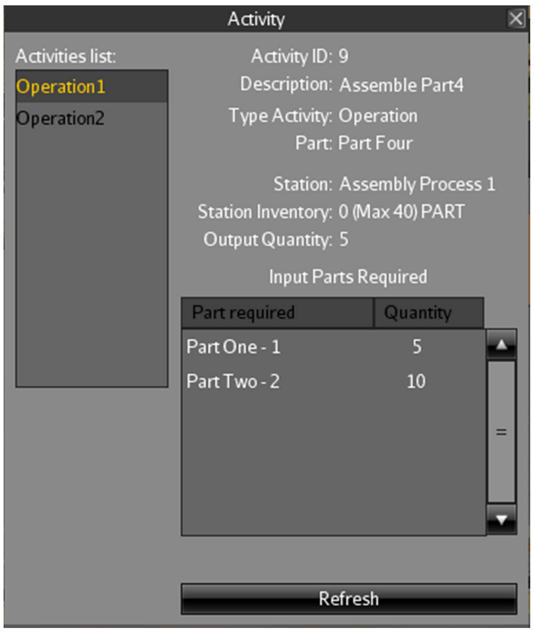
Ship	Ship Part2	
Transport	Part2 from StorageFG to Shipping	12
Transport	Part2 from Assemble to StorageFG	5
Operation	Assemble Part2	
Transport	Part1 from StorageRM to Assemble	10
Transport	Part1 from Purchase to StorageRM	10
Purchase	Purchase Part1	

Figure 6-21: Unit load screen

6.6.4. Activities Screens

6.6.4.1. Operation

The activity screen (Figure 6-18) shows an operations list on the left side. Once the user clicks on an activity, the activity's data is loaded on the right side of the screen, which provides information about the current activity and allows changing the amount of pieces to be assembled or cut. Furthermore, it has a list in the bottom of the right side, this list shows the number of parts required in order to produce a new product or part.



Activity ID: 9
Description: Assemble Part4
Type Activity: Operation
Part: Part Four
Station: Assembly Process 1
Station Inventory: 0 (Max 40) PART
Output Quantity: 5

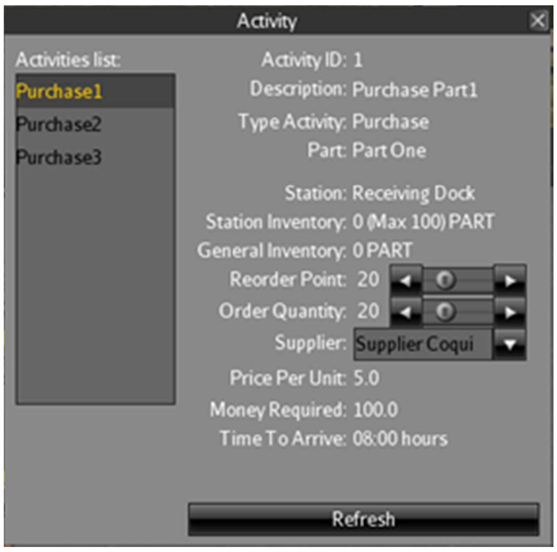
Input Parts Required

Part required	Quantity
Part One - 1	5
Part Two - 2	10

Figure 6-22: Operation activity screen

6.6.4.2. Purchase

The left side of this screen is similar than the screen in figure 6-18, the only difference is that it shows a list of purchase activities. The right side has information about the purchase activity and three (3) controls, which allow the user to set up the reorder point, order quantity, and desired supplier (Figure 6-19).


Activity

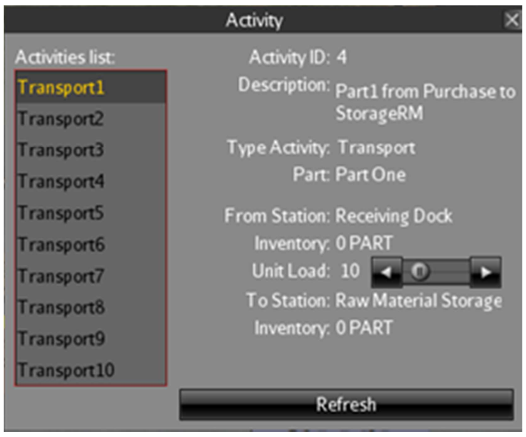
Activities list:
- Purchase1
- Purchase2
- Purchase3

Activity ID: 1
Description: Purchase Part1
Type Activity: Purchase
Part: Part One
Station: Receiving Dock
Station Inventory: 0 (Max 100) PART
General Inventory: 0 PART
Reorder Point: 20
Order Quantity: 20
Supplier: Supplier Coqui
Price Per Unit: 5.0
Money Required: 100.0
Time To Arrive: 08:00 hours

Figure 6-23: Purchase activity screen

6.6.4.3. Transport

The left side of Figure 6-20 is the same as the screen in Figure 6-19, however, the right sideshows information about the transport activity, e.g. description, part, initial station, and final station. Also, it allows the user to choose the amount of parts desired for the activity. The "refresh" button commits the changes done only to the current activity.


Activity

Activities list:
- Transport1
- Transport2
- Transport3
- Transport4
- Transport5
- Transport6
- Transport7
- Transport8
- Transport9
- Transport10

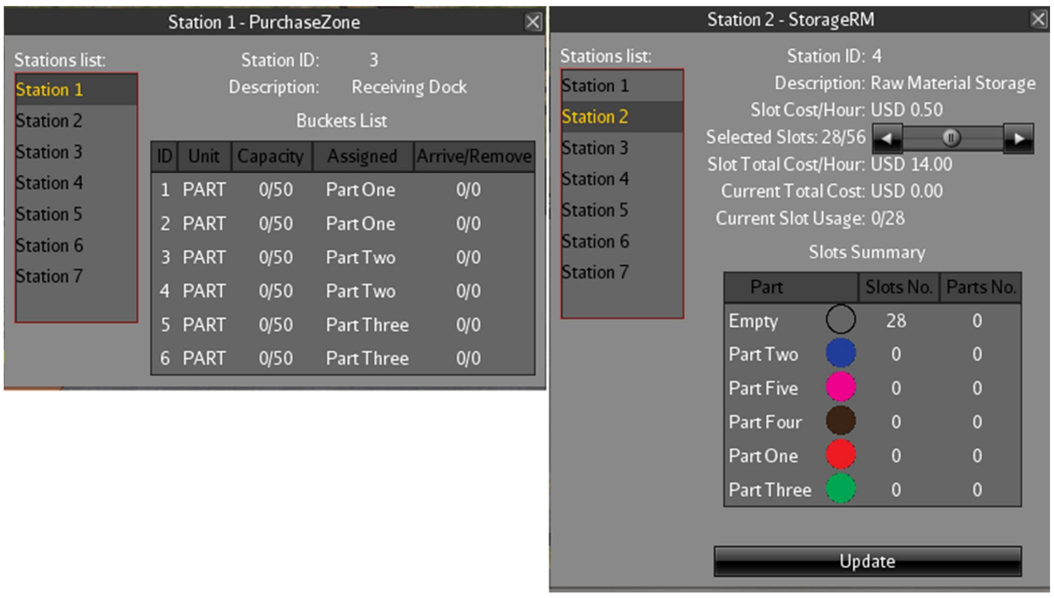
Activity ID: 4
Description: Part1 from Purchase to StorageRM
Type Activity: Transport
Part: Part One
From Station: Receiving Dock
Inventory: 0 PART
Unit Load: 10
To Station: Raw Material Storage
Inventory: 0 PART

Figure 6-24: Transport activity screen

6.6.5. Utilities Screens

6.6.5.1. Station

This game manages two (2) different kinds of stations. Figure 6-21 shows a no-storage station (left side), and a storage station (right side). The station on the left side contains a list of buckets, providing details like unit part, inventory and capacity, part assigned, among others. The storage station (right) indicates costs, which is why it provides the user a control to choose the number of slots available. Also, it has a slots summary, detailing the number of slots available, number of slots occupied for each part, and number of parts stored into slots.



Station 1 - PurchaseZone

Stations list: Station 1, Station 2, Station 3, Station 4, Station 5, Station 6, Station 7

Station ID: 3

Description: Receiving Dock

Buckets List

ID	Unit	Capacity	Assigned	Arrive/Remove
1	PART	0/50	Part One	0/0
2	PART	0/50	Part One	0/0
3	PART	0/50	Part Two	0/0
4	PART	0/50	Part Two	0/0
5	PART	0/50	Part Three	0/0
6	PART	0/50	Part Three	0/0

Station 2 - StorageRM

Stations list: Station 1, Station 2, Station 3, Station 4, Station 5, Station 6, Station 7

Station ID: 4

Description: Raw Material Storage

Slot Cost/Hour: USD 0.50

Selected Slots: 28/56

Slot Total Cost/Hour: USD 14.00

Current Total Cost: USD 0.00

Current Slot Usage: 0/28

Slots Summary

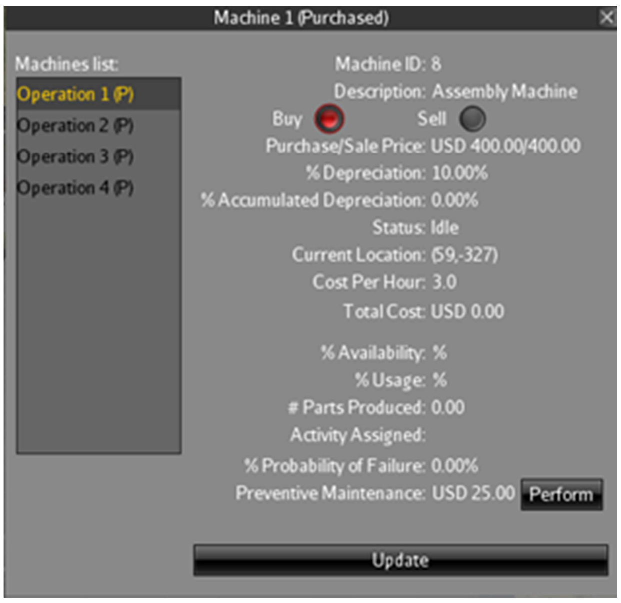
Part		Slots No.	Parts No.
Empty	○	28	0
Part Two	●	0	0
Part Five	●	0	0
Part Four	●	0	0
Part One	●	0	0
Part Three	●	0	0

Figure 6-25: Different stations: no-storage (left) and storage (right)

6.6.5.2. Machine / Equipment

The equipment and machine screens (Figure 6-22) are similar; the difference is the concept of each one. These panel enables the user to buy or sell the machine or equipment.

When the user wants to sell a machine or equipment, he/she will notice that the current sale price will decreasing. This depreciation occurs when the machine or equipment is used; also the screen shows information about regarding the depreciation, like percentage of depreciation, and percentage of accumulated depreciation. In addition, the machine or equipment has a cost per hour, percentage of availability, percentage of usage, number of parts produced (only machines), and a percentage of failure. However, if the user wants to avoid the machine to break, then he/she can perform a preventive maintenance as shown in the figure below.

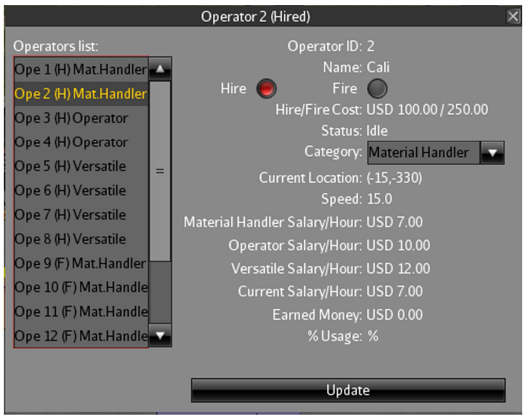


```
Machine 1 (Purchased)

Machines list:
Operation 1 (P)
Operation 2 (P)
Operation 3 (P)
Operation 4 (P)

Machine ID: 8
Description: Assembly Machine
Buy ●   Sell ○
Purchase/Sale Price: USD 400.00/400.00
% Depreciation: 10.00%
% Accumulated Depreciation: 0.00%
Status: Idle
Current Location: (59,-327)
Cost Per Hour: 3.0
Total Cost: USD 0.00
% Availability: %
% Usage: %
# Parts Produced: 0.00
Activity Assigned:
% Probability of Failure: 0.00%
Preventive Maintenance: USD 25.00 [Perform]

[Update]
```

Figure 6-26: Machine screen

6.6.5.3. Operator

Figure 6-23 shows the “operator” screen that allows the user to hire or fire the selected operator. This action should be taken carefully because hiring or firing an operator implies costs. Also, the user can change the role or category between material handler, line operator, or versatile. Finally, the screen indicates the payment per hour of each category, the full wages earned by the operator, and the percentage of usage.


Operator 2 (Hired)

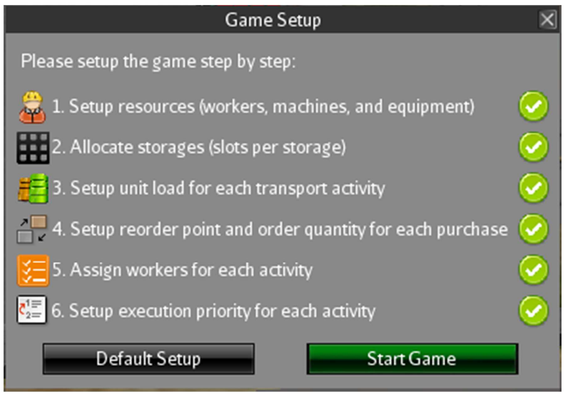
Operators list:
Ope 1 (H) Mat.Handler
Ope 2 (H) Mat.Handler
Ope 3 (H) Operator
Ope 4 (H) Operator
Ope 5 (H) Versatile
Ope 6 (H) Versatile
Ope 7 (H) Versatile
Ope 8 (H) Versatile
Ope 9 (F) Mat.Handler
Ope 10 (F) Mat.Handle
Ope 11 (F) Mat.Handle
Ope 12 (F) Mat.Handle

Operator ID: 2
Name: Cali
Hire ● Fire ○
Hire/Fire Cost: USD 100.00 / 250.00
Status: Idle
Category: Material Handler
Current Location: (-15,-330)
Speed: 15.0
Material Handler Salary/Hour: USD 7.00
Operator Salary/Hour: USD 10.00
Versatile Salary/Hour: USD 12.00
Current Salary/Hour: USD 7.00
Earned Money: USD 0.00
% Usage: %

Figure 6-27: Operator screen

6.6.6. Additional Screen

This last screen is the first one before playing the game because if the player do not set up the game, then it will be impossible to start it. Figure 6-23 shows the screen that requires a configuration step by step. Otherwise, the player should click on the “Default Setup” button and every step will be setup automatically.


Game Setup

Please setup the game step by step:

1. Setup resources (workers, machines, and equipment) ✓
2. Allocate storages (slots per storage) ✓
3. Setup unit load for each transport activity ✓
4. Setup reorder point and order quantity for each purchase ✓
5. Assign workers for each activity ✓
6. Setup execution priority for each activity ✓

Figure 6-28: Game setup screen

7. SOFTWARE INTEGRATION AND TESTING

7.1. System Overview

Software testing is the final stage in the software development life cycle, and its purpose is to evaluate, discover game defects (bugs) and fix them, in order to achieve a reliable and high quality product. Also, it helps confirm whether the game is following the specified requirements or not, and solve issues when is necessary. Among the tests available, only the “unit test” [23] and “integration test” will be used to ensure the proper functioning of the game features.

During the development phase (carried out by the author) different tests as mentioned above were done. The testing process was performed each time that the developer completed programming a feature. Once all features were completed, eight (8) testers were in charge of the testing process, using the test cases that are shown below. The test team was comprised of men and women, including gamers and non-gamers. Testing took two (2) months, and initially two developers were involved in testing for additional support. Hence, when a bug was found, or a feature improvement was approved, the development team was notified in order to deal with the issue.

7.2. Test Plan

The test plan is a list of the game’s features and each aspect of their functionality which was evaluated. Also, it defines every tool required to carry out the execution, and how the test environment is composed. During the execution, every bug found and each wrong behavior of the feature performed were solved. Once the testing is completed, then it starts again as a finite cycle, finalizing when everything is correct.

7.2.1. Features to be tested

The testing process of each feature, it means the test case, is detailed in the Appendix A. The list of features tested is shown below.

1. Logging user
2. Create new user
3. Update existing user
4. Forgotten password
5. Synchronization process
6. Game navigation
7. Factory navigation
8. Game setup function
9. Control screens
10. Activities screens
11. Utilities screens
12. Complete game level
13. Miscellaneous
14. Multiplatform compatibility

7.2.2. Features not to be tested

This section mentions different features that are not tested because they are out of the project's scope. Table 7-2 describes each feature and explains why they are not tested.

Table 7-1: Features not tested

Feature	Description	Reason why feature is not tested
PC requirements	Appendix B provides the computer requirements to execute the game.	It will depend on the player's computer. During testing, PC requirements were validated. Once completed, the game has run satisfactorily.
Synchronization speed	The synchronization process could take some seconds or minutes, depending on the amount of data and the transfer speed.	As was mentioned in the feature description, it will depend of the transfer speed which in turn depends on the user's bandwidth.
Other OS compatibility	This game release is available for the following operating systems: Windows, Linux, and Mac OS.	The game has not been tested in Unix, or any tablets or smartphones.

7.2.3. Testing Tools and Environment

Initially, when the game was in the development stage, the testing process was done manually in the same machine, without requiring any third party software to execute it. The game was tested by test team after the game development was completed. During testing, the author showed and explained all functionalities of the game, and each tester received the game installer that was installed in their computers. These computers complied with the PC requirements, as shown in Appendix B.

Furthermore, testing took place once a week, for different purposes, for example: detailed results and feedback were given to the development team, new game versions were tested and new tasks were executed. In addition, testers did not use any automated testing mechanism, everything was done manually and stored in a Wiki page for the game.

8. CONCLUSIONS AND FUTURE WORK

8.1. Conclusions

Game development has been a challenge for the team, given the game complexity, as well as the use and integration of different emerging technologies. The current version provides 4 different levels of increasing difficulty and can be easily installed in different platforms, i.e. Windows, Mac OS and Linux.

Currently the game has completed the pilot phase (testing process), and is ready to be played by freshman students, in order to evaluate the research objective which is to know if the students learned or not Industrial Engineering core concepts and improved problem-solving skills. As part of this testing process, many critical bugs have been fixed but there are still others of lower priority that must be addressed. Furthermore, the GUI design has been improved in order to make the user's interaction as simple and efficient as possible.

Finally, we expect that this project will promote the development of other educational activities based on gaming, given the lack of Industrial Engineering research through games.

8.2. Future Works

The game development was completed in accordance with the estimated scope. However, some improvements and changes are still being considered to be implemented for a future game release.

8.2.1. Graphics Designs and Animation Improvements

Currently, this game release was focused on developing a product that allows students to learn through gaming, giving less emphasis on the design and animation of graphics objects. For that reason, a future release of the game will contain:

- Operators with better design.
- Machines with better quality graphics.
- Raw materials and products will represent real parts of the factory
- The animation of an operator who is handling equipment will be more realistic.
- The animation when an operator is picking and placing parts on the equipment will be more realistic.
- Fonts and screen style will be interchangeable, allowing the user to set up as desired.

8.2.2. More Advance Game Levels

Given that the game loads each level from the database, it allows the administrator to add many more levels from the remote database. Once the player starts the application, then the engine will synchronize both databases, getting these new levels from the remote database. Moreover, these levels can allow the user to continue learning and figure out more strategies to successfully complete each level.

8.2.3. Allow User to Create and Share New Levels

Considering the advantage to add and update game levels from the database, it would be a great achievement creating a drag-and-drop tool that allows designing new levels by the player. This tool should be easy of usage and must have an option to share this new game level with other players. This good practice is common on the biggest games as StarCraft [29] and Age of Empires 3 [30], and promotes to create a community of gamers.

8.2.4. Game Migration to Android Devices

The game was developed in Java and tested in personal (laptop) and desktop computers. However, the game migration to Android devices does not demand much effort as if it were created from scratch. Also, this migration process will have to consider updating the GUI and graphic design, simplifying the game engine given that Android devices do not provide many resources, and managing the local database in the synchronization process. Finally, the most adequate devices to play this game are tablets, given the screen size, processors and memory capacity.

REFERENCES

[1] Emrich, A. (2004-2005). The gamer generation and why baby boomers shouldn't worry so much about them http://www.alanemrich.com/SGI/Week_10/SGI%2010%20GAMER%20GENERATION.pdf

[2] Thilmany, J. (2012). Gaming Pros and Cons. Mechanical Engineering; Mar2012, Vol. 134 Issue 3, p20-20

[3] Peterson, J. L. (1981), Petri net theory and the modeling of systems, 1st ed. Englewood Cliffs, N.J.: Prentice-Hall Inc., pp. x+290

[4] IEEE Std. 830-1998 IEEE Recommended Practice for Software Requirements Specifications

[5] IEEE Std. 1016-1998 IEEE Recommended Practice for Software Design Descriptions

[6] IEEE Std. 829-1998 IEEE Standard for Software Test Documentation

[7] IEEE Std. 1008-1997 IEEE Standard for Software Unit Testing

[8] IEEE Std. 1012-1998 IEEE Standard for Software Verification and Validation

[9] Java SE 7 – Oracle Technology Network (2012), http://www.oracle.com/technetwork/java/javase/overview/index.html

[10] JMonkeyEngine 3.0. Java OpenGL Game Engine (2012), http://jmonkeyengine.com/engine/

[11] Nifty GUI 1.3.1 http://nifty-gui.lessvoid.com/

[12] MySQL – Database and Workbench http://www.mysql.com/

[13] SQLite – Database engine http://www.sqlite.org/

[14] OpenGL – Open Graphics Library – The Industry's Foundation for High Performance Graphics. http://www.opengl.org/

[15] SimPack by Dr. Paul Fishwick – University of Florida http://www.cise.ufl.edu/~fishwick/Welcome.html

[16] Hart, P., Nilsson, N., Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems and Cybernetics SSC4; 100-107

[17] Kelly, H., Howell, K., Gilnert, E., Holding L., Swain, C., Burrowbridge, A., Roper, M. (2007). How to Build Serious Games.
Communications of the ACM; Jul2007, Vol.50 Issue 7, p44-49

[18] Mayo, Merrilea J. (2007). Games for Science and Engineering Education. Communications of the ACM; Jul2007, Vol.50 Issue 7, p30-35

[19] Bernardi, G., Cesta, A., Coraci, L., Cortellessa, G., De Benedictis, R., Mohier, F., Polutnik, J. (2011). Only hope remains in the Pandora's. http://www.pandoraproject.eu/

[20] Zapusek, M. (2011). Serious computer games as instructional technology. Mipro, 2011 Proceedings of the 34th International Convention

[21] Eric Freeman, Elisabeth Freeman, Kathy Sierra and Bert Bates (2004). Head First Design Patterns, First Edition, O'Reilly Media, Inc.

[22] OMG (2007), OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF

[23] IEEE Std. 1008-1987 IEEE Standard for Software Unit Testing: An American National Standard

[24] Unity 3D v4.1.2
http://www.unity3d.com/unity

[25] UDK 3.0 (Unreal Development Kit)
http://www.unrealengine.com/udk

[26] GameStart – Cross-Platform 3D & 2D Engine – Beta 2R7
http://www.gamestart3d.com/home

[27] Cocos2d 0.5.5
http://www.cocos2d.org/index.html

[28] Carl Adam Petri and Wolfgang Reisig (2008) Petri Net. Scholarpedia http://www.scholarpedia.org/article/Petri_net

[29] StarCraft 2
http://us.battle.net/sc2/en/game/maps-and-mods/

[30] Age of Empires 3
http://www.ageofempires3.com/

APPENDIX A

Test Cases

1. Logging User

Each time a user wants to play, the game will require a user and password. Otherwise, it will be impossible start the game.

Table 1: Logging user

Test Case ID	Test Case Name	Description	Expected Results	Comments
1.1	Validate fields	User and password must be correct.	No error messages.	Password is case-sensitive.
1.2	Login successful	Once fields are validated the game is accessed.	Access the game.	Synchronization process of user data begins.
1.3	Login failed	If user and/or password are incorrect or not registered.	A detailed error message is shown.	

2. Create New User

A new player must create a new user in the system. In order to do that, the game will need an Internet connection to validate the new user with existing user data in the remote database. Once the player completes the registration and it is validated, then he/she will receive the generated password in the email registered.

Table 2: Create new user

Test Case ID	Test Case Name	Description	Expected Results	Comments
2.1	Access to the screen successfully	Clicking on “create new user” button will retrieve the appropriate screen.	Access to the “create new user” screen.	Displayed fields are empty.
2.2	Validate fields	Fields are validated to check that they have been correctly filled.	If correct, it will proceed to the registration.	When it is not validated, it will show an error message.
2.3	Register successfully	Once fields are validated, it will register the new user.	It sends the generated password to the email.	The player will not be able to play if he/she adds a fake email.

3. Update Existing User

After the player logged into the system, he or she will be in the main menu. Clicking the “Profile” button changes the screen and shows user profile, enabling fields to be modified. Finally, the “Update Account” button will return the player to the main menu.

Table 3: Update existing user

Test Case ID	Test Case Name	Description	Expected Results	Comments
3.1	Access to the screen successfully	Clicking the “profile” button will enter the appropriate screen.	It will load the user data in fields.	The registered email cannot be changed.
3.2	Validate fields	Fields are validated to check that they have been correctly filled. Also, the old	If correct, it will proceed to the update.	When it is not validated, it will show an error message.

		and new password field.		
3.3	Update successfully	Once fields are validated, it will update the user data.	The updated data will be stored locally and remotely.	The new password will be required for the next session.

4. Forgot your Password

In case the registered user forgot his/her password, then the player should click on the "Forgot Your Password?" link. The game will change to another screen, and it will ask to enter the registered email. Finally, the user should receive an email with his/her password.

Table 4: Forgotten password

Test Case ID	Test Case Name	Description	Expected Results	Comments
4.1	Access to the screen successfully	Clicking the "forgot your password" link will enter the appropriate screen.	It will show the email field.	The field is empty.
4.2	Validate field	It will validate the email entered.	Once it is validated then it will send the password to the email.	The email should be registered
4.3	Validate receiving email	The game will send a message to the email account	The message will contain the recovered password	The password should be the indicated by the user

5. Synchronization Process

It occurs twice: when the user executes the program and when the user logs into the game. It means that the game instance synchronizes the game and user information stored with the remote database. It requires Internet connection; otherwise, it will not work.

Table 5: Game data synchronization

Test Case ID	Test Case Name	Description	Expected Results	Comments
5.1	Validate game data synchronization	It synchronizes the game data between the remote server and the user machine executing the game.	Game data updated.	The user machine must have an Internet connection.
5.2	Validate user data synchronization	It synchronizes the user data between the remote server and the user machine executing the game.	User data updated.	The user machine must have an Internet connection.

6. Game Navigation

This feature validates the correct consistency when the user clicks on different buttons in the main menu, and the game responds accurate. For example: if the user clicks the “Credits” button, the game should show a screen displaying game credits.

Table 6: Game Navigation

Test Case ID	Test Case Name	Description	Expected Results	Comments
6.1	Validate	Clicking this button	It shows the screen	

	“New game” button click	will access another screen which contains the list of games to be played.	with the list of games available.	
6.2	Validate “Profile” button click	Clicking this button will access the “profile” screen.	It shows this screen with the current user data.	
6.3	Validate “Tutorial” button click	Clicking this button will load the tutorial in a browser.	Internet browser containing the tutorial.	It will open the default Internet browser.
6.4	Validate “Return to game” button click	As the button’s description says, clicking on it will return to the game.	It will return to the game only when a game is running.	When no game is running, then this button is disabled.
6.5	Validate “Credits” button click	The button will open a pop-up that will show game credits.	Credits will be shown in a pop-up.	This information identifies the main author and crew.
6.6	Validate “Switch user” button click	The button will logout the current user, and it will return to the login screen.	Once the action is completed it will show the login screen.	Fields in the login screen are empty.
6.7	Validate “Quit game” button click	It will trigger a pop-up asking if the user wants to exit the game or not.	Pop-up asks the player to confirm closing the game.	The player can cancel the pop-up and return to game.

7. Factory Navigation

This feature occurs in the game environment, i.e. the factory, and it is triggered each time that the user clicks and holds right button on the mouse, then the game should move the camera at the same direction that the mouse is moved.

Table 7: Factory navigation

Test	Test Case	Description	Expected Results	Comments

Case ID	Name			
7.1	Validate camera navigation	It allows rotating (i.e. 360 degrees) and zooming the game's camera.	The game’s camera has zooming and rotation limit.	This activity can be performed using the right button or center button of the mouse.

8. Game Setup Function

This feature occurs mostly at the beginning of each game level. In the “Game Setup” screen, it shows six (6) steps to set up before starting a game level. Also, it has a button called “Default Setup”; it is used in case the player does not want to waste time configuring the game. However, this functionality validates when the user clicks on the image of each step in the game setup screen to open the appropriate screen.

Table 8: Game setup function

Test Case ID	Test Case Name	Description	Expected Results	Comments
8.1	Validate “Start game” before setup	When at least one step is not setup then the game will not start.	The game engine does not allow starting the game.	The play button located in the top left of the screen does not allow the game to start either.
8.2	Validate “Start game” after setup	Every step must be setup to allow the game to start.	It starts the game after it was correctly validated.	The screen closes after the action.
8.3	Validate click on each setup button	Each image that identifies the step or each red circle image is going to open an appropriate screen.	Open a screen according to the step selected.	Once a step is set up, then the red circle image is updated to a green circle image.

9. Control screens

The feature's objective is to validate the consistency of the user's action with the game. Once the player clicks the "Control" button, the game will show other five (5) buttons: allocate storages, assign operators, priority activities, resources, and unit load. Each one will display a screen with some functionality.

Table 9: Control screens

Test Case ID	Test Case Name	Description	Expected Results	Comments
9.1	Validate actions on "storages" screen	Once this screen opens, it shows a list of storages. The player will be able to set up some options.	See the actions taken in the game, e.g. added slots per storage.	The game has two different storages. The first one is read-only information. The other one allows changes.
9.2	Validate actions on "assign operators" screen	It allows allocating and deallocating operators for each activity.	Operators perform some activities during the game.	It allows choosing operators.
9.3	Validate actions on "priority activities" screen	It will allow choosing the priority of execution for each activity.	The game executes activities by priority.	It will not allow choosing the same priority for more than one activity.
9.4	Validate actions on "resources" screen	It will allow hiring and firing operators, and buying and selling machines or equipment.	The number of operators, machines and equipment can vary during the game.	A fired operator or sold machine is identified by a red sphere on the top.

9.5	Validate actions on “unit load” screen	It allows the player to choose the amount of parts or raw material to move during a transport activity.	The game is going to move the amount selected for each transport activity.	The amount selected cannot be less than zero.

10. Activities screens

This feature also validates the consistency of user actions. Clicking the “Activities” button shows: operation, purchase, and transport buttons. Furthermore, these buttons display an appropriate screen with some functions. Clicks on buttons or screens will be validated

Table 10: Activities screens

Test Case ID	Test Case Name	Description	Expected Results	Comments
10.1	Validate actions on the “operation” screen	This screen shows the list of operations available for this game level. The player can change the amount of parts to be produced.	The machine is going to produce the amount of parts selected for each operation.	The operation time will vary depending on the amount of parts to be produced.
10.2	Validate actions on the “purchase” screen	Screen shows the list of purchases available for this game level. The player can change the reorder point and order raw material, and select a supplier.	Raw material could take more time to arrive, and the cost varies depending of the supplier selected.	Note that this activity will be executed automatically each time the inventory is less than the reorder point.
10.3	Validate actions on	The screen shows the list of transports	The game is going to use the unit load for	

	the "transport" screen	available. The player will be able to change the unit load for each activity.	each transport activity.	

11. Utilities screens

Clicking the "Utilities" button will show: station, machine, equipment, operator, part, and supplier buttons. They will open different screens to be validated, and will look for consistency.

Table 11: Utilities screens

Test Case ID	Test Case Name	Description	Expected Results	Comments
11.1	Validate actions on the "station" screen	This screen shows the list of stations available for this game level. The player will be able to change some options.	Actions taken by the player in storages will be shown during the game.	Stations different than storages, are read-only.
11.2	Validate actions on the "machine" and the "equipment" screen	It will allow to buy or sell machines/equipment as desired by the player.	Buying or selling machines/equipment will be shown during the game, it means in the factory.	The player can sell a machine or equipment that is working, but the action is carried out when the machine is idle.
11.3	Validate actions on "part" and "supplier" screen	It will allow the user to choose between parts or suppliers. This information is read-only.	The game will show the information according to what the user has selected.	

12. Complete Game Level

This feature validates that the game will finish after some orders have arrived, winning or losing the game level, or falling in bankruptcy. The completed game will show a pop-up with statistical information and provide some options to the player.

Table 12: Complete game level

Test Case ID	Test Case Name	Description	Expected Results	Comments
12.1	Validate pop-up shown	Game has three different ways to finish the level. First, falling in bankruptcy; second, failing more orders than allowed; finally, receiving all orders.	It will show the pop-up, once the game is completed.	The player is not able to make any changes after the pop-up is shown.
12.2	Verify won game	The user will win the game when he/she completes more orders than required and a pop-up will be shown.	Game will show a pop-up saying that the player won, and will provide statistical information.	
12.3	Verify lost game	The user will lose the game when he/she fails more orders than permitted or falls in bankruptcy, and a pop-up will be shown.	Game will show a pop-up saying that the player has lost, and will provide statistical information.	

13. Miscellaneous

This feature validates that other controls are working correctly. For example: game time, next order due timer, next purchase timer, play button, and pause button.

Table 13: Miscellaneous

Test Case ID	Test Case Name	Description	Expected Results	Comments
13.1	Verify game time and slider control	It will validate the relation between the speed selected in the slider and the current game time.	The game time should forward faster if the slider has increased the speed.	Graphic objects are also affected with the speed of the slider control.
13.2	Validate next order due timer	This timer shows the remaining time complete an order.	If the user does not fulfill the order before the timer counts zero, then order is lost.	The timer shows the nearest order to be lost, when the game has two or more orders to be attended.
13.3	Validate next purchase timer	This timer shows the remaining time to receive more raw materials, according to the purchase activity.	The player will receive more raw materials when timer is zero.	This time will vary depending on the supplier selected.
13.4	Verify play and pause button action	Once the game is running, these buttons are available.	The game enables these options after the initial setup has been completed.	If the game is paused and the player clicks on pause, then this action will not have any effect.

14. Multiplatform Compatibility

This feature should validate the correct execution of the game in different platforms (i.e. Windows, Linux, Mac OS), having previously installed the game.

Table 14: Multiplatform compatibility

Test Case ID	Test Case Name	Description	Expected Results	Comments
14.1	Verify installation and execution on Windows	Game development provides a Windows installer.	Game is installed and runs successfully.	Game runs from Windows XP to Windows 8
14.2	Verify installation and execution on Mac OS	Game development provides a Mac OS installer.	Game is installed and runs successfully.	It has been tested in Mac OS version 10.6 and latest.
14.3	Verify installation and execution on Linux	Game development provides a Linux installer.	Game is installed and runs successfully.	It has been tested in Linux Ubuntu, Fedora, and Mint.

APPENDIX B

System Requirements and Game Installation

Minimum System Requirements:

- OS: Windows XP/Vista/7/8 with DirectX 9.0. Mac OS X 10.6.8 or newer. Linux Ubuntu/Fedora/Mint.
- Processor: Intel Pentium D 2.8 GHz or AMD Athlon 64 X2 4400 or better.
- Graphic card: Intel HD Graphics 3000 or NVIDIA GeForce G210 or ATI Radeon HD 5450 or better.
- Hard disk: 1 GB available or more.
- RAM memory: 1 GB (1.5GB required for Windows users, 2GB required for Mac users) or more.
- Others: broadband Internet connection, and 1280x780 minimum display resolutions.

Recommended Specifications:

- OS: Windows 7/8. Mac OS X 10.7 or newer. Linux Ubuntu/Fedora/Mint (latest version).
- Processor: Intel Core 2 Duo 2.4 GHz or AMD Athlon 64 X2 5600 2.8 GHz or better.
- Graphic card: NVIDIA GeForce 260 or ATI Radeon HD 4870 or better.
- Hard disk: 2 GB available or more.
- RAM memory: 2 GB

Game Installation:

1. Download the appropriate installer for your operating system.
2. Start the installation:
    a. Windows: double-click on the installer.
    b. Mac OS: double-click on the installer.
    c. Linux: open a Terminal, type “sh” then add the full path of the installer location plus the installer name file.
3. Once started, click on the “Next” button, then “read and accept” the user agreement.
4. Click on the “Next” button and then choose the folder location of the game.
5. In the new screen, click on the “Next” button, and finally it will copy the files into your computer.
6. Finally, click on the “Close” button.

The user will find the game installed in his/her programs list, and the game will be ready to play.

GLOSSARY OF TERMS

- API – Application programming interface is a set of protocols, routines, and tools for building software applications.
- GPU – Graphics processing unit, it is used primarily for 3D applications.
- OpenGL – It is a 3D graphics language that was built into Windows NT and is designed to improve performance on hardware that supports this standard.
- SDK – Software development kit. It is a programming package that enables a programmer to develop applications for a specific platform.
- RC2 – Release candidate 2. It is a beta version with potential to be a final product.
- GUI – Graphical user interface. This user interface allows user to interact with the computer or specific software.
- XML – Extensible markup language. It allows developers to create their own customized tags, enabling the definition, transmission, validation and interpretation of data between applications.
- RDBMS – Relational database management system. It means that stores data in the form of related tables.
- SMTP – Simple mail transfer protocol. It is a protocol for sending email messages.
- TCP/IP – Transmission control protocol / Internet protocol. It is the suite of communications protocols used to connect hosts on the Internet.
- IEEE – Institute of Electrical and Electronics Engineers. It is best known for developing standards for the computer and electronics industry.