



UNIVERSITY OF BEIRA INTERIOR
Faculty de Engineering
Departament of Informatics

Support Tools for 3D Game Creation

Pedro Nuno Matos Pereira

Dissertation submitted in candidature for the
Degree of Master of Science in Informatics Engineering

(2nd cycle degree)

Supervised: Prof. Doutor Frutuoso Silva

Department of Informatics
University of Beira Interior
Covilhã, Portugal
<http://www.di.ubi.pt>

Acknowledgements

First of all, I would like to thank to Professor *Frutuoso Silva* for all the support, for the constant words of encouragement and for supervising my Master's Thesis. I also would like to thank for the opportunity to belong to his research group: Reality, Games and Graphics Group (*ReGain*).

Additionally, I'd like to thank to all members of Clean World development team, *André Barbosa*, *David Casteleira* and *João Dias* for all the support and help.

Last but not least, I also like to all my friends, particularly to *Carolina Belino*, *David Massano*, *Eduardo Fonseca*, *Laurie Geerlings*, *Luis Rodrigues*, *Marco Antunes*, *Nélia Fonseca* and *Sofia Colmier* for all the support, help and good times we had in the last years. In one-way or another, all of you have contributed to the accomplishment oh this dissertation.

Resumo Alargado

Atualmente, as ferramentas para o desenvolvimento de jogos são uma parte bastante importante de todo o processo de desenvolvimento. Estas ferramentas servem para assistir os criadores de jogos nas tarefas que realizam, permitindo-lhes a criação de jogos funcionais escrevendo poucas linhas de código. Desenvolver um videojogo sem a utilização de ferramentas especializadas é um processo complexo e que consome bastante tempo, daí a existência de ferramentas que permitem ao utilizador importar os conteúdos para o jogo, definir a lógica de jogo, produzir o código fonte e compilá-lo. Este tipo de software é normalmente utilizado por quem se dedica à criação de jogos como hobby, ou por profissionais que procuram otimizar o processo de desenvolvimento de jogos.

Existem várias componentes ao nível do desenvolvimento de videojogos que se tornam pouco produtivas, se não forem automatizadas e/ou otimizadas. Por exemplo, a programação de eventos ou de diálogos pode ser uma tarefa que consome demasiado tempo no ciclo de desenvolvimento, para além de ser uma tarefa entediante e repetitiva no ponto de vista do programador. Por este motivo, a utilização de ferramentas pode ser muito importante no que diz respeito ao aumento da produtividade e manutenção dos vários processos que envolvem o desenvolvimento de videojogos. Nesta dissertação pretendemos demonstrar as vantagens da utilização dessas mesmas ferramentas durante o desenvolvimento de videojogos, através da apresentação de um caso de

estudo que envolve o desenvolvimento de um *Serious Game* intitulado *Clean World*.

Em *Clean World*, foram identificadas determinadas tarefas que se mostraram demasiado repetitivas e entediantes quando programadas por inteiro, como é o caso da adição, modificação ou remoção de componentes como diálogos, *quest* ou *items*. Tendo em conta este problema concreto, foram criadas algumas ferramentas de forma a aumentar a produtividade no desenvolvimento do jogo, tornando tarefas repetitivas e entediantes em processos simples e intuitivos. O conjunto de ferramentas é constituído por: *Item Manager*, *Quest Manager*, *Dialog Manager* e *Terrain Creator*.

Abstract

Nowadays, tools for developing videogames are a very important part of the development process in the game industry. Such tools are used to assist game developers in their tasks, allowing them to create functional games while writing a few lines of code. For example, these tools allow the users to import the content for the game, set the game logic, or produce the source code and compile it.

There are several tasks and components regarding the development of videogames that may become unproductive, therefore, it's necessary to automate and/or optimize such tasks. For example, the programming of events or dialogs can be a task that consumes too much time in the development cycle, and a tedious and repetitive task for the programmer. For this reason, the use of tools to support these tasks can be very important to increase productivity and help on the maintenance of the various processes that involve the development of videogames. This dissertation aims to demonstrate the advantages of the use of these kind of tools during the development of videogames, presenting a case study involving the development of a Serious Game entitled Clean World.

Keywords

Game Development Support Applications, XNA, 3D Games, Serious Games.

Contents

Acknowledgements	i
Resumo Alargado	iii
Abstract.....	v
Keywords.....	vii
Contents	ix
List of Figures.....	xi
List of Tables	xv
Acronyms.....	xvii
1. Introduction.....	1
1.1. Focus	1
1.2. Objectives	2
1.3. Main Results.....	2
1.4. Dissertation Structure	3
2. Related Work.....	4
2.1. Introduction.....	4
2.2. Video Game Development	6
2.3. Game Engines and Frameworks	9
2.4. Game Development Tools	12
2.4.1 . Game Maker	13
2.4.2. The 3D Gamemaker	15
2.4.3. CopperCube	16
2.4.4. 3D Rad.....	18
2.4.5. RPG Maker.....	19
2.4.6. NeoAxis Game Engine	23
2.4.7. Unity 3D	28
2.4.8. UDK	35

2.5. Summary	49
3. Clean World – The 3D Game.....	50
3.1. Introduction	50
3.2. Game Storyline.....	52
3.3. Modeling and Animation of the Characters	54
3.3.1. Boris	54
3.3.2. Kate	67
3.3.3. Dr. Jacob	69
3.3.4. Tom A. Toe.....	72
3.4. Terrain Models and Level Assets.....	74
3.4.1. Terrain Models	74
3.4.2. Decorative 3D Assets	76
3.4.3. Interactive and Collectable 3D Assets	83
3.5. Level Design	87
3.6. 2D Assets	91
3.7. Summary	97
4. Developed Support Applications	99
4.1. Introduction	99
4.2. Item Manager	100
4.3. Quest Manager	106
4.4. Dialog Manager	111
4.5. Terrain Creator	116
4.6. Connection with Clean World	120
4.7. Summary	121
5. Conclusions and Future Work.....	122
5.1. Conclusions	122
5.2. Future Work.....	123
References.....	125

List of Figures

<i>Fig 2. 1 - A: 1952, OXO/Noughts and Crosses (Tic-Tac-Toe); B: 1958: Tennis for Two.....</i>	<i>5</i>
<i>Fig 2. 2 – Screenshot from the game BioShock.....</i>	<i>7</i>
<i>Fig 2. 3 – Screenshot from the interface of Game Maker.</i>	<i>14</i>
<i>Fig 2. 4 – Screenshot from the interface of The 3D Gamemaker.....</i>	<i>16</i>
<i>Fig 2. 5 – Screenshot from the interface of CopperCube.....</i>	<i>17</i>
<i>Fig 2. 6 – Screenshot from the interface of 3D Rad.</i>	<i>18</i>
<i>Fig 2. 7 – Screenshot from the interface of RPG Maker 2000.</i>	<i>20</i>
<i>Fig 2. 8 – A: Screenshot from the battle system of RPG Maker 2003; B: Screenshot from the battle system of RPG Maker VX.....</i>	<i>20</i>
<i>Fig 2. 9 – Screenshot from the script editor of RPG Maker XP.</i>	<i>21</i>
<i>Fig 2. 10 – Screenshot from the interface of RPG Maker VX.</i>	<i>22</i>
<i>Fig 2. 11 – Game class hierarchy used by NeoAxis Engine.....</i>	<i>24</i>
<i>Fig 2. 12 – A screenshot from the NeoAxis Engine map editor tool.</i>	<i>25</i>
<i>Fig 2. 13 – A screenshot from the NeoAxis Game Object Editor.....</i>	<i>26</i>
<i>Fig 2. 14 – A screenshot from Unity light mapping editor.</i>	<i>29</i>
<i>Fig 2. 15 – A screenshot from Unity Editor interface.</i>	<i>32</i>
<i>Fig 2. 16 – A screenshot from Unity editor grid feature.</i>	<i>33</i>
<i>Fig 2. 17 – A screenshot from Shuriken.....</i>	<i>34</i>
<i>Fig 2. 18 – UDK structure.....</i>	<i>36</i>
<i>Fig 2. 19 – Screenshot from the UDK editor interface.</i>	<i>37</i>
<i>Fig 2. 20 – Screenshot from the UDK Matinee editor interface.</i>	<i>38</i>
<i>Fig 2. 21 – Screenshot from the UDK Kismet interface.</i>	<i>39</i>
<i>Fig 2. 22 – Screenshot from the UDK Material editor interface.</i>	<i>40</i>
<i>Fig 2. 23 – Screenshot from the UDK Sound editor interface.</i>	<i>41</i>
<i>Fig 2. 24 – Screenshot from the UDK Cascade editor interface.</i>	<i>42</i>
<i>Fig 2. 25 – Screenshot from the UDK Lightmass editor interface.....</i>	<i>43</i>
<i>Fig 2. 26 – Screenshot from the UDK Terrain tools interface.....</i>	<i>44</i>
<i>Fig 2. 27 – Screenshot from the UDK Cinematic tools interface.....</i>	<i>44</i>
<i>Fig 2. 28 – Screenshot from the UDK Fracture Meshes editor interface.</i>	<i>45</i>
<i>Fig 2. 29 – Screenshot from the UDK SpeedTree editor interface.....</i>	<i>46</i>
<i>Fig 2. 30 – Screenshot from the UDK FaceFX editor interface.</i>	<i>47</i>
<i>Fig 2. 31 – Screenshot from the Microsoft Visual Studio editing UnrealScript.....</i>	<i>48</i>
<i>Fig 3. 1 - Screenshot from the game Clean World.....</i>	<i>51</i>
<i>Fig 3. 2 – Boris, the hero from the game.</i>	<i>55</i>
<i>Fig 3. 3 – Model sheet of Boris in robot mode.</i>	<i>55</i>
<i>Fig 3. 4 – Model of Boris in robot mode in Autodesk 3Ds Max 2012.....</i>	<i>56</i>

<i>Fig 3. 5 – UVW map of the model of Boris in robot mode.....</i>	<i>57</i>
<i>Fig 3. 6 – UVW map of the model of Boris in robot mode with textures.....</i>	<i>57</i>
<i>Fig 3. 7 - A: Model of Boris without textures; B: Model of Boris with textures.....</i>	<i>58</i>
<i>Fig 3. 8 - Model of Boris with biped inserted.....</i>	<i>58</i>
<i>Fig 3. 9 – Adjusting the bones envelopes to the 3D model of Boris.....</i>	<i>59</i>
<i>Fig 3. 10 - Model of Boris in sphere mode in Autodesk 3Ds Max 2012.....</i>	<i>61</i>
<i>Fig 3. 11 - A: UVW map of the model of Boris in sphere mode; B: UVW map of the model of Boris in sphere mode with textures.....</i>	<i>62</i>
<i>Fig 3. 12 - A: Model of Boris without textures; B: Model of Boris with textures.....</i>	<i>62</i>
<i>Fig 3. 13 - Model of Boris in sphere with bone inserted.....</i>	<i>63</i>
<i>Fig 3. 14 - Model of Boris in solar mode in Autodesk 3Ds Max 2012.....</i>	<i>64</i>
<i>Fig 3. 15 - A: UVW map of the model of Boris in solar mode; B: UVW map of the model of Boris in solar mode with textures.....</i>	<i>64</i>
<i>Fig 3. 16 - A: Model of Boris without textures; B: Model of Boris with textures.....</i>	<i>65</i>
<i>Fig 3. 17 - Model of Boris in solar mode with biped inserted.....</i>	<i>65</i>
<i>Fig 3. 18 - A: Normal map of Boris in robot mode; B: Normal map of Boris in sphere mode; C: Normal map of Boris in solar mode.....</i>	<i>66</i>
<i>Fig 3. 19 – Concept of Kate given by the concept artist.....</i>	<i>67</i>
<i>Fig 3. 20 – Model sheet of the character Kate.....</i>	<i>67</i>
<i>Fig 3. 21 - A: Model of Kate without textures; B: Model of Kate with textures.....</i>	<i>68</i>
<i>Fig 3. 22 - A: Model of Kate with biped inserted; B: Adjusting the envelopes of Kate model.....</i>	<i>68</i>
<i>Fig 3. 23 – Concept of Dr. Jacob given by the concept artist.....</i>	<i>69</i>
<i>Fig 3. 24 – Model sheet of the character Dr. Jacob.....</i>	<i>70</i>
<i>Fig 3. 25 - A: Model of Dr. Jacob without textures; B: Model of Dr. Jacob with textures.....</i>	<i>70</i>
<i>Fig 3. 26 - A: Model of Dr. Jacob with biped inserted; B: Adjusting the envelopes of Dr. Jacob model.....</i>	<i>71</i>
<i>Fig 3. 27 – Concept of Tom A. Toe given by the concept artist.....</i>	<i>72</i>
<i>Fig 3. 28 – Model sheet of the character Tom A. Toe.....</i>	<i>72</i>
<i>Fig 3. 29 - A: Model of Tom A. Toe without textures; B: Model of Tom A. Toe with textures.....</i>	<i>73</i>
<i>Fig 3. 30 - A: Model of Tom with biped inserted; B: Adjusting the envelopes of Tom model.....</i>	<i>73</i>
<i>Fig 3. 31 – Complete model of the Cypricene Island.....</i>	<i>75</i>
<i>Fig 3. 32 – A: Terrain of level one; B: Terrain of level three.....</i>	<i>75</i>
<i>Fig 3. 33 – A: Terrain of level two.....</i>	<i>76</i>
<i>Fig 3. 34 – kate’s house model.....</i>	<i>77</i>
<i>Fig 3. 35 – Lamp ring model.....</i>	<i>77</i>
<i>Fig 3. 36 – Rails model.....</i>	<i>77</i>
<i>Fig 3. 37 – Medical center model.....</i>	<i>78</i>
<i>Fig 3. 38 – Wood recycling machine model.....</i>	<i>78</i>
<i>Fig 3. 39 – Toxic waste cleaner model.....</i>	<i>78</i>
<i>Fig 3. 40 – Wind tower model.....</i>	<i>79</i>
<i>Fig 3. 41 – Lighthouse model.....</i>	<i>79</i>
<i>Fig 3. 42 – Wall model.....</i>	<i>79</i>
<i>Fig 3. 43 – Platform model.....</i>	<i>80</i>
<i>Fig 3. 44 – Part of the bridge model.....</i>	<i>80</i>
<i>Fig 3. 45 – Island harbor model.....</i>	<i>80</i>
<i>Fig 3. 46 – Energy pole model.....</i>	<i>81</i>
<i>Fig 3. 47 – Watermill tower model.....</i>	<i>81</i>
<i>Fig 3. 48 – Tunnel section model.....</i>	<i>81</i>
<i>Fig 3. 49 – Mining walker model.....</i>	<i>82</i>

<i>Fig 3. 50 – Watermill model.....</i>	<i>82</i>
<i>Fig 3. 51 – WWTP model.....</i>	<i>82</i>
<i>Fig 3. 52 – Tom’s house model.</i>	<i>83</i>
<i>Fig 3. 53 – Question mark model.</i>	<i>83</i>
<i>Fig 3. 54 – Recycling machine model.</i>	<i>84</i>
<i>Fig 3. 55 – Yellow recycling container model.</i>	<i>84</i>
<i>Fig 3. 56 – Arrow model.</i>	<i>84</i>
<i>Fig 3. 57 – Recycling platform model.....</i>	<i>85</i>
<i>Fig 3. 58 – Solar platform model.</i>	<i>85</i>
<i>Fig 3. 59 – Recharger model.....</i>	<i>85</i>
<i>Fig 3. 60 – Console terminal model.....</i>	<i>86</i>
<i>Fig 3. 61 – The syringe collectable item model.</i>	<i>87</i>
<i>Fig 3. 62 – Positioning a 3D model on SunBurn editor.</i>	<i>88</i>
<i>Fig 3. 63 – Changing the color of the ambient light.</i>	<i>89</i>
<i>Fig 3. 64 – Changing the color of a directional light.....</i>	<i>89</i>
<i>Fig 3. 65 – Changing the settings of a point lighth.....</i>	<i>90</i>
<i>Fig 3. 66 – A: Normal map used on metallic surfaces; B: Kate’s house without normal map; C: Kate’s house with normal map.....</i>	<i>91</i>
<i>Fig 3. 67 – A: Main menu background; B: Game over menu background.....</i>	<i>92</i>
<i>Fig 3. 68 – Example of a loading screen used in the game.....</i>	<i>92</i>
<i>Fig 3. 69 – Health and energy bars sprite.....</i>	<i>93</i>
<i>Fig 3. 70 – Boris power mode indicator on its several forms.....</i>	<i>93</i>
<i>Fig 3. 71 – Mini map used on level one.</i>	<i>93</i>
<i>Fig 3. 72 – Example of the communicator used in the game.</i>	<i>94</i>
<i>Fig 3. 73 – Example of a dialog sprite used in the game.....</i>	<i>94</i>
<i>Fig 3. 74 – A: Background of the items tab; B: Background of the load tab.</i>	<i>95</i>
<i>Fig 3. 75 – Example of a quest sprite used in the game.</i>	<i>95</i>
<i>Fig 3. 76 – Background of one of the developed mini games.....</i>	<i>96</i>
<i>Fig 3. 77 – Tutorial screen of the first bonus level.</i>	<i>96</i>
<i>Fig 4. 1 – Screenshot of the Item Manager tool.....</i>	<i>100</i>
<i>Fig 4. 2 – Creating a new item on Item Manager tool.</i>	<i>101</i>
<i>Fig 4. 3 – A: Item information on Item Manager; B: Item information loaded into the game.</i>	<i>102</i>
<i>Fig 4. 4 – Screenshot of the error message showing-up.....</i>	<i>102</i>
<i>Fig 4. 5 - Screenshot of a new inserted item.</i>	<i>103</i>
<i>Fig 4. 6 – A: Selecting an item from the list; B: Modifying the item data.</i>	<i>103</i>
<i>Fig 4. 7 – Opening a saved file with the application.....</i>	<i>104</i>
<i>Fig 4. 8 – Screenshot of the Quest Manager tool.....</i>	<i>106</i>
<i>Fig 4. 9 – Creating a new quest on Quest Manager tool.</i>	<i>107</i>
<i>Fig 4. 10 – A: Quest information on Quest Manager; B: Quest information loaded into the game..</i>	<i>108</i>
<i>Fig 4. 11 - Screenshot of the error message showing-up.....</i>	<i>108</i>
<i>Fig 4. 12 - Screenshot of a new inserted quest.....</i>	<i>109</i>
<i>Fig 4. 13 – A: Selecting a quest from the list; B: Modifying the quest data.</i>	<i>109</i>
<i>Fig 4. 14 – Opening a saved file with the application.</i>	<i>110</i>
<i>Fig 4. 15 – Screenshot of the Dialog Manager tool.</i>	<i>111</i>
<i>Fig 4. 16 – Creating a new dialog on Dialog Manager tool.....</i>	<i>112</i>
<i>Fig 4. 17 – A: Dialog information on Dialog Manager; B: Dialog information loaded into the game.</i>	<i>113</i>

<i>Fig 4. 18 - Screenshot of the error message showing-up.....</i>	<i>113</i>
<i>Fig 4. 19 - Screenshot of a new inserted dialog.....</i>	<i>114</i>
<i>Fig 4. 20 – A: Selecting a dialog from the list; B: Modifying the dialog data.....</i>	<i>114</i>
<i>Fig 4. 21 – Screenshot of the Terrain Creator tool.....</i>	<i>116</i>
<i>Fig 4. 22 – Painting the heightmap with different pen sizes and height levels.</i>	<i>117</i>
<i>Fig 4. 23 – Erasing the heightmap with the eraser tool.</i>	<i>117</i>
<i>Fig 4. 24 – Loading texture files to the terrain.....</i>	<i>118</i>
<i>Fig 4. 25 – Terrain successfully created.</i>	<i>119</i>
<i>Fig 4. 26 – Created terrain on Autodesk 3Ds Max.....</i>	<i>120</i>

List of Tables

<i>Table 1 - Companies that developed titles using RPG Maker.</i>	<i>23</i>
<i>Table 2 - Companies that developed titles using NeoAxis Engine.....</i>	<i>27</i>
<i>Table 3 - Companies that developed titles using Unity 3D Engine.</i>	<i>35</i>
<i>Table 4 - Companies that developed titles using Unreal Engine.</i>	<i>48</i>
<i>Table 5 – Clean World development team.</i>	<i>51</i>

Acronyms

AI	:	Artificial Intelligence
CAD	:	Computer-Aided Design
DLL	:	Dynamic Link Library
FPS	:	First Person Shooter
GML	:	Game Maker Language
GUI	:	Graphic User Interface
IDE	:	Integrated Development Environment
LOD	:	Level of Detail
RPG	:	Role Playing Game
RTS	:	Real-Time Strategy
TPS	:	Third Person Shooter

1. Introduction

1.1. Focus

Presently, specific software tools for game creation are a very important part of the game development process. These tools enable users to create functional games, with little or none programming. The user should be able to import the assets to the game, define the game logic, and behaviors he desires to see. In the end, these software tools should be able to produce the source code of the game, compile it, and create a full and playable game, based on the instructions given by the user.

This kind of software has two main applications:

- usage by hobbyists with little or none knowledge in programming;
- usage by professionals in order to increase productivity in the game development process;

Nowadays, we have several examples of this kind of tools being used both by professionals and hobbyists. One example of a professional tool is the UDK [1] from the Epic Game [2]. This tool allows users to define the game logic and rules, materials, animations, particle behaviors, among other possibilities, without programming a single line of code. Such powerful tools make the game development process and the deployment easier. There are also several examples of this kind of software for hobbyists use, like the RPG Game Maker [3], or the Game Maker [4]. However, these tools don't have the power or complexity of the UDK, being simpler and friendlier to the user.

This dissertation presents a 3D game, as well as some tools to improve the game development process. These software tools allow the game developers to produce content for the games, with a simple interface and at the same time the power of a fully functional 3D game engine.

1.2. Objectives

The main objective of this dissertation is the creation of software tools to improve the creation of 3D games. Besides, the author collaborated in the development of a 3D game which participated in Microsoft Imagine Cup 2012, on the game design category for Xbox360 and PC. This game was developed in XNA and the software tools were created to improve the development process of this game.

To accomplish this main objective, the following partial objectives were identified and performed:

1. Creation a story of the Clean World game;
2. Creation of the 2D and 3D content of the game;
3. The development of software tools to improve some tasks of the development process of the Clean World game;
4. Evaluation of the developed tools.

1.3. Main Results

This section is devoted to present the main results obtained in the scope of this dissertation.

The first result was the participation in the Microsoft Imagine Cup 2012, in the Xbox/PC category, with Clean World, that was classified in 7th place on the worldwide finals. Note that our team was the first Portuguese team reaching the worldwide finals in game design.

The second result was the software tools developed to improve the

process of game creation. There tools are also presented in a paper submitted to Videogames 2012. Besides, a demo of Clean World was also submitted to Videogames 2012.

1.4. Dissertation Structure

This dissertation is organized in 5 chapters. This chapter, the first, presents the context of the dissertation, focusing on the topic under study, the objectives, the contributions and presents the dissertation structure.

Chapter 2 elaborates on the related work about the topic, focusing on the existing software tools used nowadays, both by the game industry and by the hobbyists.

Chapter 3 presents the contribution to the creation of a fully functional 3D game (Clean World), as well as the creation of all game contents.

Chapter 4 focuses on the development of specific software tools to improve the development process of Clean World, as well as the creation of the game content using these tools.

Finally, Chapter 5 presents some conclusions, and points some directions for future research works.

2. Related Work

2.1. Introduction

Games are almost as old as Man itself, the relationship between this two started millenniums ago. The “Royal Game of Ur”, which dates from the middle millennium B.C., is the oldest board game known until today. Other example of how ancient is this relationship between Man and games is the “Baoying-Xiangqi”, the oldest chess game in History [5].

Initially, the number of exiting games as very low. The development process was very slow, since they were handcrafted, which created several difficulties in manufacture. The few that existed, also had difficulties spreading among societies, since the communication among people that time was hard [6]. However, with the Industrial Revolution, this reality changed. With the use of machines was now possible to do a large-scale manufacturing of games. Due to this large-scale production, many game like Monopoly (1903) and Detective (1947) were later launched, having much success among the communities.

However, the reign of board games didn’t last forever. In 1952 and 1958, with the creation of the first computer game (OXO) and videogame (Tennis for Two) respectively (see figure 2.1), a new era started, the era of electronic games. During the 80s, these new forms of digital entertainment started to took over the market, and the board games lost ground to the digital games.

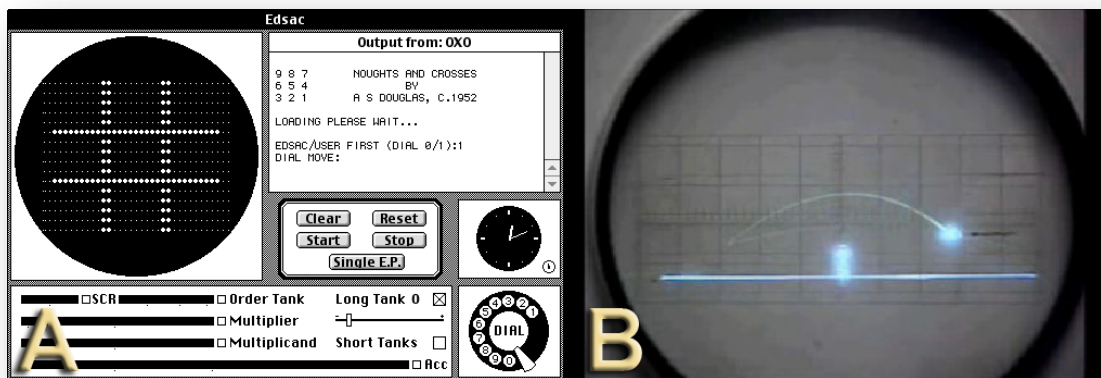


Fig 2. 1 - A: 1952, OXO/Noughts and Crosses (Tic-Tac-Toe); B: 1958: Tennis for Two.

The market of digital games is a growing sector, moving a volume of 11.7 billion dollars in the year of 2008 only in the USA. The growth of this industry surpassed the 200% in the past 10 years [7], making the game industry an essential part of the entertainment industries.

We can tell how successful a game is by how much it enthralls his user. Development companies struggle to produce games that capture the players' attention in such way, that their perception of time and sense of self becomes distorted or forgotten [8].

In the last three decades, the consumer video game hardware has evolved from simple 8-bit processors, some dedicated display logic and a few kilobytes of memory to high clock rate multi-core processors coupled with a programmable graphics unit, large memories, high definition audio/video output, and sophisticated user input devices. This tremendous increase in capability led to an increase in the sophistication of games. Since the need to create complex projects has risen so greatly, the project team sizes are now over hundred people, instead of a couple programmers [9].

This evolution has drawn the game development from basements, where it all started years ago, to multi-national companies that created a business that rivals with Hollywood. Unfortunately today, the computer hobbyist that started the creation of electronic games find too difficult to

create their own games, mostly because they want to create something similar to what is done in the industry, but the task is too complex.

2.2. Video Game Development

The development of video games projects has several characteristics that differ from typical projects of software development. For start, games are software that has the purpose of entertaining, they have multimedia contents that are similar to those used in movies. That's the reason why today game industry uses multi-disciplinary teams and has the need to adapt itself to the constant changes in the market, having to assimilate quickly the new tendencies and technological innovations. Besides, the complexity and specificity of the game usually dictates the way this one is produced [10].

Diverse assets such as 3D models, textures, animations, sound, music, dialog, video and other resources are integrated in the game by the development pipeline. These multimedia assets are created by specialists that work together with the programmers in order to create the code framework in which all assets fits.

During the development of the game, several tools can be used, such as software engines for physics or third party software for modeling and character animation. Sometimes it's even needed to code some plug-ins for this other specialized tools, in order to import the assets into the game [11].

An example of how massive a video game project can be is *BioShock* [12] (see figure 2.2). It took three years to develop, employed 93 in-house developers, 30 contractors, and 8 on-site publisher testers. In the end, the final result was a total of 3,775 files for the game, 758,903 line of native C++ code, and 184,144 lines of script code.



Fig 2. 2 - Screenshot from the game *BioShock*.

So, in order to consider some of the unusual factors that are not present in traditional software development, the called multi-disciplinary teams were created. However, the team organization varies from company to company. These teams are usually segregated by specialty, for example, a programming group and a design group. These groups can have sub-groups, such as an AI team or a texture team. A common practice is to put an experienced employee in the area in charge of one team.

Still, some companies are adopting a more agile style and have broken down the traditional groups to create functional units. An example of this can be a unit composed by two programmers, one texture artist and one animator [11].

Some typical designations that are used to refer each member of a game development team are the following [10]:

- **Producer:** Is the one in charge of planning the game and follow the development process. The producer works both internally and externally, trying to create a connection between the development

team and the clients/sponsors of the game. The producer should have a macro vision of the project, getting involved in all its aspects.

- **Artist:** Is the one who creates all art material for the game. There are several specializations, such as animator, or 2D artist. The artist helps to define the visual identity of the game, the conception of the characters, the textures, the 3D modeling, etc.
- **Game Designer:** Is the one responsible for creating the artist conception of the game, in a similar way to what is done in a movie. This activity requires great knowledge about games in the market. It's also he's job to make sure that all pieces of the game are put together in a coherent way, so the final product can be fun and changeling.
- **Programmer:** The programmer is the one responsible for creating the source code of the game. They can do several activities related to game programming, like creating the code needed for render, the physics programming, the code for the sound system, the code for the AI, etc.
- **Sound Engineer:** Is the one that creates the audio material for the game, both sound effects and soundtrack. He works with a team of artists in other to create the audio-visual identity of the game, based on the info given by the game designer.
- **Quality Engineer:** His job is to make sure that the final game has enough quality for the standards of the market. He makes tests to the source code of the game and also to the *gameplay*. This role has been getting more and more significance due to the need to reduce the number of flaws in software, in order to increase quality.

2.3. Game Engines and Frameworks

During the game development, according to the design and implementation, the production of the game can become highly expensive [13]. In order to change this situation, new approaches were used in game development, using available game engines and frameworks [14] [15].

In the past, games were developed line by line using a programming language or a scripting language. Using this tedious process means that the code used in games was not able to be reused [16]. The fast development of the computer technology made this technique over past, nowadays there is a new form to design games: the game engines.

Game engines work for games just like a car engine works for a car [17]. Like in cars, based on the engine, we can produce different models with different body style and colors. The same happens with game engines, allowing us to create a variety of games with different characters, scenes and stories.

Using game engines makes the task of design games easier, since we no longer have to design the game from scratch. Game engines are frameworks for games, that after inputting codes about storyline, setting control information, the sound, the 3D assets, etc., a game can be created. However, the choice of the engine is important in order to carry out level design, animation design and modeling [18]. It's also important to consider that most engines are turned into a particular content style. For example, if we use a game engine tuned to flight simulators, we may have poor results using it in a game placed in tunnels and dungeons [19].

Usually, we can define a 3D game engine the following way: It's a mix of several engines that allow the programmer to manage the game assets, like sound and music, but also gives power to the programmer to control the game artificial intelligence, the game physics, or the collision detection system. We can define its outputs as graphics, music, and sound effects,

and the input is given by the input device, which can be a controller, a keyboard, or even a sensor. It provides the algorithms to perform several tasks, like making a character move, control the topography of the terrain, manage the artificial intelligence of the game NPCs (Non Player Characters), or even support and monitor the network [20].

A 3D game engine has several components, let's look at the function of each component briefly:

- **Rendering Engine:** The rendering engine represents the object in the screen. The two major libraries available for PC are DirectX by Microsoft, and OpenGL by Silicon Graphics Inc.
- **Animation Engine:** Is the engine responsible for all character animation in the game. The majority of the engines use limp *ragdolls* for animations, however, some new animation engines like Euphoria [21], employ more complex methods to animate the entirety of physically bound objects within the game environment.
- **Physics Engine:** It provides an approximate simulation of certain physical systems, for example: rigid body dynamics, including collision detection; soft body dynamics; and fluid dynamics. We can separate the physics engine into two categories: real-time and high-precision. High-precision is usually used in animated movies, since it requires more processing power. On the other hand, real-time physics engines are used in computer games and other forms of interactive computing since simplified calculations and decreased accuracy are used to compute in time for the game to respond at an appropriate rate for gameplay.
- **Artificial Intelligence Engine:** It's the responsible for creating the illusion of intelligence in the game. This engine controls the NPCs (Non Player Characters), and defines their behavior during the gameplay. It's also the responsible for the implantation of path

finding algorithms, which consist in methods for determining how to get an NPC from one point on a map to another.

- **Network Engine:** The network engine is the one that manages the network in online games. Makes each user contact the server sharing one space and interaction based on network. It usually includes a set of algorithms for communications between players, the possibility to create rooms and matches, and also manages the game updates.
- **3D Sound Engine:** The 3D sound engine generates the game sound data to the game progress state. This audio technology uses several techniques to simulate depth in sounds. With stereophonic audio systems, where sounds could only be heard from the left or right, there's no depth notion. The 3D sound engine focus in positing sounds in three dimensions, making the game experience more realistic.

Some game engines possess tools for map editing and animation; however, most of these tools are complex for newcomers to use. Nevertheless, some game engines don't have a graphic interface, which means that all the work like putting a 3D model in a specific location must be done by code.

2.4. Game Development Tools

We will refer to game development tools to software build over game engines that allow users to create their own game with little knowledge of programming. Typically, this software present an interface based on drag and drop operations, giving the user an intuitive and simple way for creating games.

Some of the software even allows the more advanced users to do some programming using simple script language. However, the modifications that they can do to the engine are usually very limited.

There are several of these tools on the market today, some of them quite simples to use, like the *Game Maker* [4] developed by *Yo Yo Games* [22], *The 3D Gamemaker* [23], developed by *The Game Creators* [24], *CopperCube* [25] from *Ambiera* [26], *3D Rad* [27] a freeware development tool, and probably the most well-known and more powerful, the *RPG Maker* [3] from *Enterbrain* [28].

However, there are also some professional engines that use a set of tools to help in the game creation. Most of these tools are more powerful than the ones mentioned earlier, and come integrated with the game engine. Each tool is responsible for a very specific task in the development process. Some examples of engines that use these kinds of tools are *NeoAxis Engine* [29], *Unity 3D* [30], and *UDK* [1].

2.4.1. Game Maker

The *Game Maker* [4] is an IDE for Windows and Mac, created by Mark Overmars in 1999. Was developed using the programming language Delphi and it is currently developed and published by *Yo Yo Games* [22].

The *Game Maker* allows users to create their own computer games without any knowledge of programming languages. However, it allows the more advanced users to create more complex games, using a build-in scripting language.

The first version of *Game Maker* was called *Animo* [31] and was a software tool specialized in 2D animation. Each release of *Game Maker* added new features and improved stability, making the software gain popularity. In 2001, the version 3.0 implemented DirectX for the first time. The *Game Maker 8* was released on 22 December 2009, and added new features such as a revamped script editor, an improved image editor, and the ability to import and export resources from game source files.

The user interface (see figure 2.3), uses a drag-and-drop system, allowing users unfamiliar with traditional programming to intuitively create games visually organizing icons on the screen. These icons represent actions that would occur in the game, such as movement, basic drawing, and control structures.

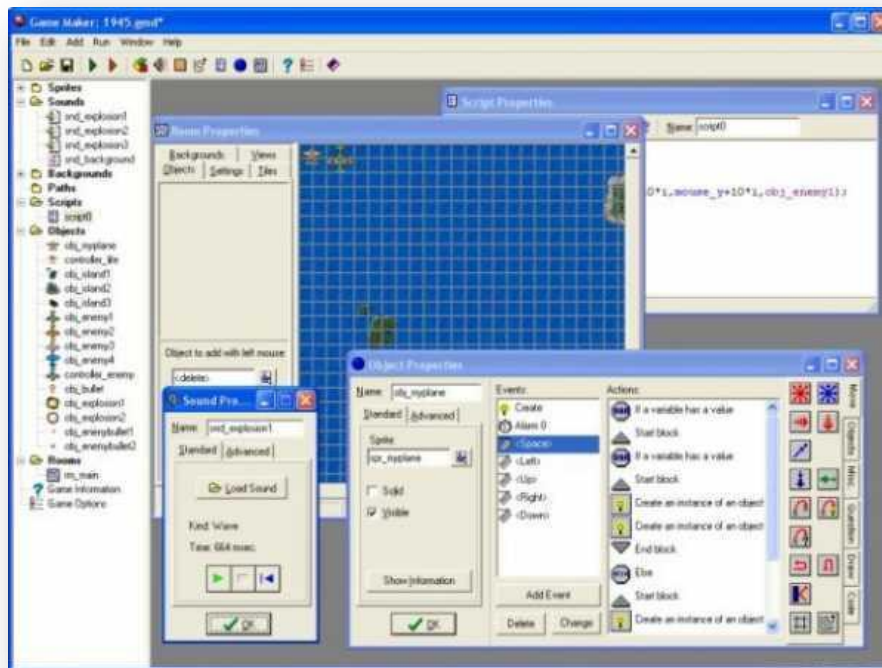


Fig 2. 3 - Screenshot from the interface of *Game Maker*.

There are two versions of *Game Maker*: the Lite and the Standard. The Lite versions are free to use, while the Standard requires purchase. The difference between these two lies on the fact that Lite locks several advanced features and functions, such as the ability to include DLLs, particle systems, advanced drawing functions, 3D graphics and network multiplayer. All these features are unlocked on the Standard version.

The *Game Maker* was primarily developed for 2D graphics, however, since version 6.0 (Windows), the software added Direct3D, allowing the use of limited 3D graphics. On the Macintosh versions 7.0 and superior, is used OpenGL. Extension packages such as *OGRE* [32] can be added to increase the 3D functionality.

The *Game Maker* uses its own scripting language, GML. This language allows advanced users to further enhance and control the design of the game through convectional programming. The syntax of GML is similar to C, C++ and Java.

2.4.2. The 3D Gamemaker

The *3D Gamemaker* [23] is a software tool developed by the company *The Game Creators* [24]. This piece of software allows users to create simple shooter/adventure games for Microsoft Windows. This tool doesn't require any programming or art knowledge, however, the users can add custom media to their creations using *DarkBASIC* [33] programming language.

Just like *The 3D Gamemaker*, *DarkBASIC* was created by the company *The Game Creators*. This programming language is a structured form of BASIC and has the purpose of game creation using Microsoft DirectX. It is faster and easier to use when compared with other languages, but it's also less powerful.

The 3D Gamemaker has a built-in editor that allows users to indicate the position of enemies, items and obstacles (see figure 2.4). Still, this option is not available on the beginner mode or lite edition. There are also other tools in this software, like a simple level creator.

This software is usually a bad choice for most users; since it has poor-quality models, very limited customization options, and the games created by the software are made from a series of pre-built parts than can be combined later on. It is more recommend for children who want to create simple games.



Fig 2. 4 - Screenshot from the interface of *The 3D Gamemaker*

2.4.3. CopperCube

CopperCube [25] is a powerful 3D engine/editor from *Ambiera* [26] (see figure 2.5), that can be used to create full functional games. This software can deploy for Flash, WebGL/JavaScript, Windows, and Mac OS X. This software is more focused on the creation 3D scenes, however, it gives the possibility of creating games by allowing the use of scripting.

The software allows the users to import their 3D assets. It supports many file formats, and since version 3.0, the editor also supports low poly editing, this means that the users can create and edit their 3D models without any third party software usually used for that task.

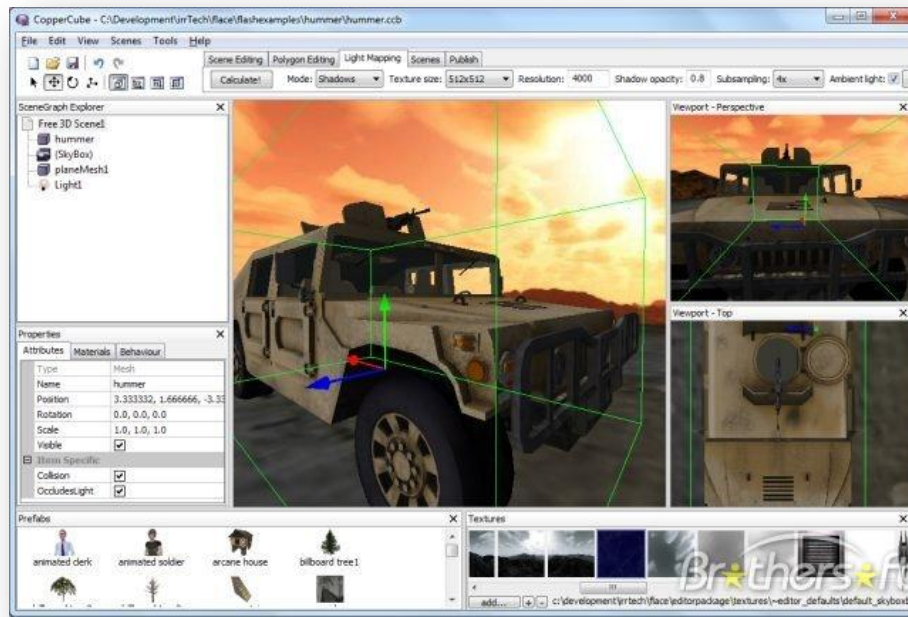


Fig 2. 5 - Screenshot from the interface of *CopperCube*.

CopperCube also supports skeletal animation for 3D characters. The meshes can have an unlimited number of joints and unlimited unmounts of weights. The 3D characters can be animated in any animation software and then imported to *CopperCube*.

The deployment on the various platforms that *CopperCube* supports is also very straight forward. With just one click of the user, the 3D scene can be deployed to any of the supported platforms. When publishing, *CopperCube* uses OpenGL, DirectX, WebGL or Stage3D/Molehill to render the scenes. There's also a *lightmapper* build in the editor that enhance the appearance of the 3D scene. This tool is very easy to use and includes features like global illumination effects.

CopperCube supports scripting, events and actions. For example, the software has a built-in event/action system. This allows the user to define actions, like playing a sound when entering a certain area, without any programming needed. If the user is publishing for Windows platform, then it's possible to use JavaScript. The same happens with Flash, allowing the user to use ActionScript 3.

CopperCube already contains some pre-created game AI behaviors and actions. Just with a click, the user can define the game characters and the enemies.

2.4.4. 3D Rad

3D Rad [27] is a freeware software used for the creation of 3D games. This software was designed for non-coders, and aims to provide the simplest development workflow possible, without sacrificing flexibility.

The *3D Rad* editor is based on a collection of components, called objects, which can be put together and configured in several ways. This allows the creation of games without coding a single line. However, for more advanced users, *3D Rad* supports scripting via *AngelScript* [34], an extremely flexible cross-platform scripting library designed to allow applications to extend their functionality through external scripts.

The editor of *3D Rad* (see figure 2.6), is almost entirely mouse-driven, making it very simple to use.

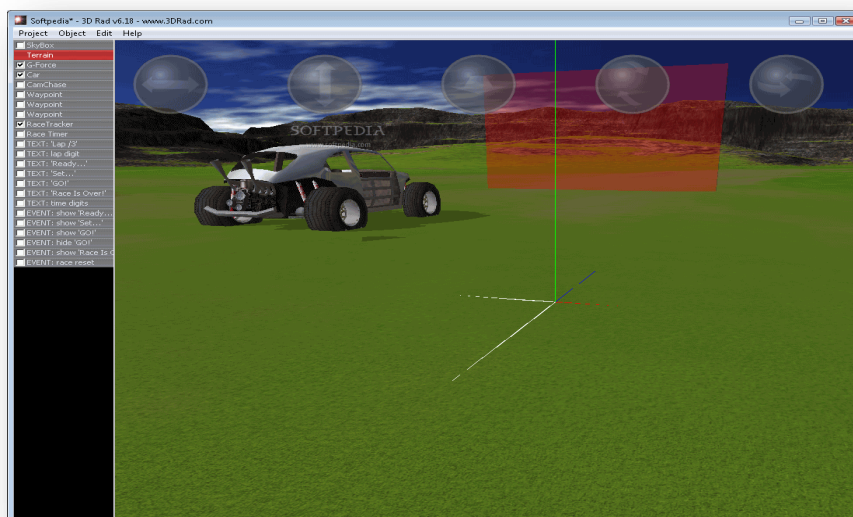


Fig 2. 6 - Screenshot from the interface of *3D Rad*.

The object types that *3D Rad* uses are based on geometry, like static/animated meshes or physics objects, can be visually combined in a

preview of the scene. Plus, object types such as forces, joints, springs and wheels can also be visually configured. Certain event-objects, like tracing detection zones, can be edited visually.

3D Rad possesses a series of visual effects such as fire, smoke or water that are already defined in the editor. Also, some post-processing effects can be created by configuring the camera.

This software supports several file formats for the assets, has two modes for games, first person and third person, and has some simple AI already configured, for example for characters, however, for more complex IA is recommended the use of scripting.

2.4.5. RPG Maker

RPG Maker [3] is probably the most successful piece of software used by hobbyists in game development. It's a series of programs for the development of 2D RPG games, created by the Japanese group *ASCII*, and then continued by *Enterbrain* [28].

In opposite to what happens with the software seen in the previous sections, *RPG Maker*, also known in Japan by the name of *RPG Tsukūru*, focus in one particular game genre, the RPG's. This software allows its users to create their own full RPG games.

RPG Tsukūru Dante 98, released on December 17, 1992, was the first of the *RPG Maker* series. Later, it was released *RPG Maker 95*, which was the first Microsoft Windows based version. Although it was early version, *RPG Maker 95* had a higher screen resolution, and higher sprite and tile resolution than some of following versions.

RPG Maker 2000 (see figure 2.7), was the second release of the software for Microsoft Windows, and is the most popular and used version so far.

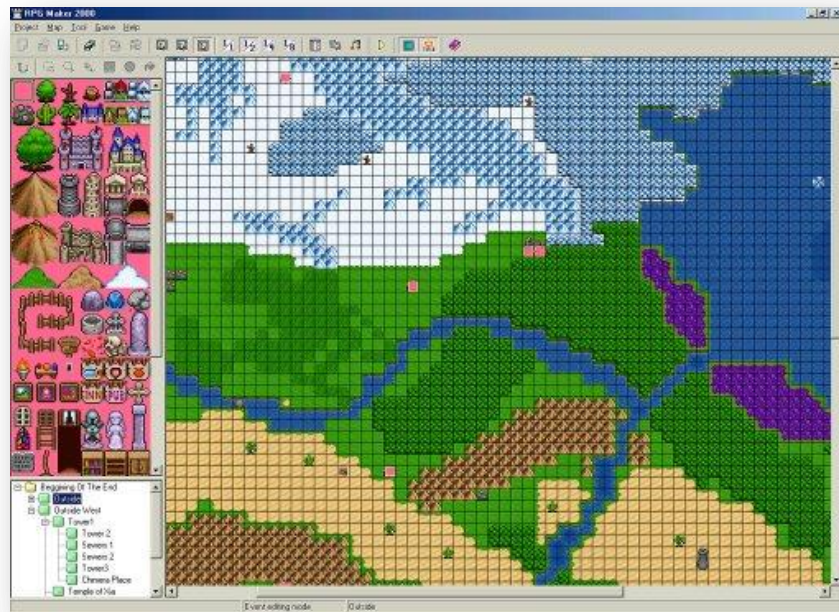


Fig 2. 7 - Screenshot from the interface of RPG Maker 2000.

The next version of the software released was *RPG Maker 2003*, this time developed by *Enterbrain*, which had previously been a part of ASCII. This version was a largely improvement of *RPG 2000*, and introduced a new battle system with side-view (see figure 2.8), similar to the one used on the classic *Final Fantasy* games.



Fig 2. 8 - A: Screenshot from the battle system of *RPG Maker 2003*; B: Screenshot from the battle system of *RPG Maker VX*.

However, the big breakthrough in the series was in *RPG Maker XP* and its successor, *RPG Maker VX*. These versions allow the more advanced users to use scripting in Ruby (see figure 2.9), making these versions very powerful. Additionally, these versions have more control over sprite sizes, since there is no longer a specific image size regulation for sprite sheets, like in previous versions. Also, some other aspects of the game design where improved.

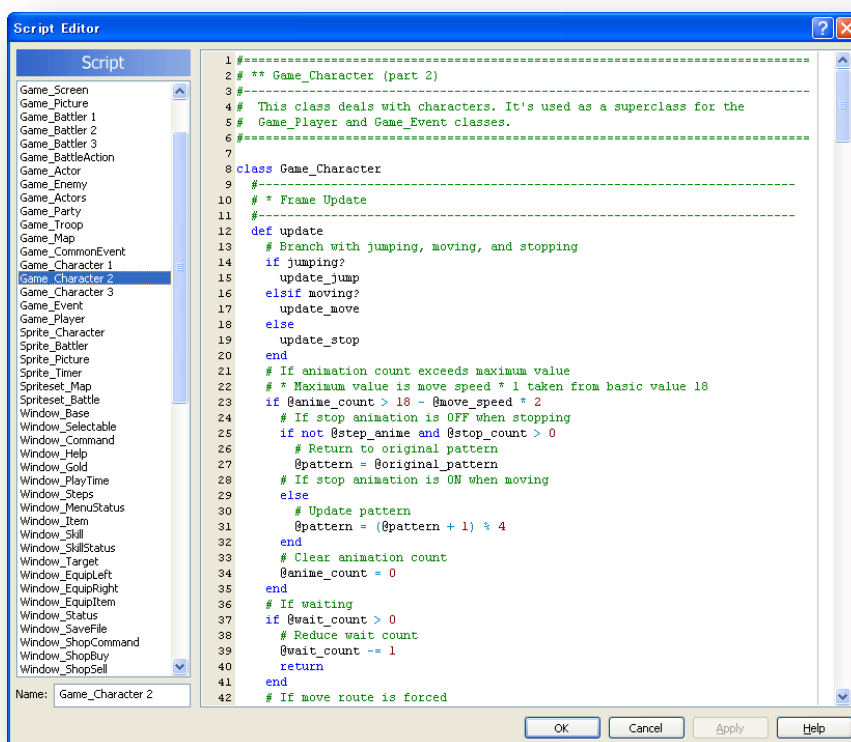


Fig 2. 9 - Screenshot from the script editor of *RPG Maker XP*.

RPG Maker series interface (see figure 2.10), is quite simple and very straight forward, even to newcomers. It includes a tile set based map editor, called chipsets in versions previous to *RPG Maker XP*, a simple system of creating events and applying the basic logic operations, and a battle editor. All versions include premade *tilesets*, characters, and events. However, the users can upload to the date base their own *tilesets* and sprites, allowing then to customize the game experience.

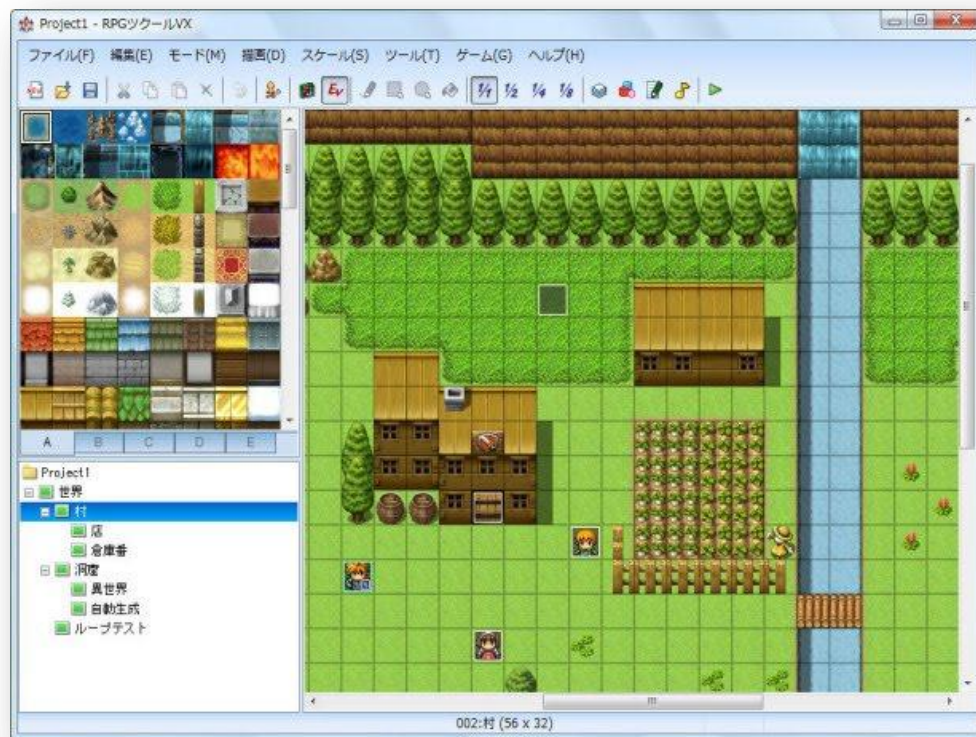


Fig 2. 10 - Screenshot from the interface of *RPG Maker VX*.

The big success of the *RPG Maker* series is no doubt related to the simplicity of the interface and the ability of completely customize the games created by this software. If this software was originally designed for non-coder hobbyists, since the software allowed the use of Ruby as scripting languages, several independent game development companies have been using *RPG Maker* to create commercial games, which are distributed by *Big Fish Games* [35], *IGN Entertainment* [36] and *GamersGate* [37]. In the following table we present some of the companies that use *RPG Maker* and some of the games created by them.

Table 1 - Companies that developed titles using *RPG Maker*.

Independent game development companies	Developed game titles
Aldorlea Games [38]	Millennium [39]
Amaranth Games [40]	Aveyond Series [41]
Blossomsoft [42]	Eternal Eden [43]
Over Cloud 9 [44]	Arevan [45]
Sherman3D [46]	Alpha Kimori [47]

2.4.6. NeoAxis Game Engine

NeoAxis Game Engine [29] is a complete game engine with all the feature of a modern 3D game engine. This software can be used to create all types of games, from casual games to AAA games. This engine can be used on Microsoft Windows together with Microsoft Visual Studio for development, and can deploy to several platforms, like Microsoft Windows XP/Vista/7, Apple Mac OS X, and Linux. In development are the deploy possibilities for Apple iOS, Google Android, and Google Native Client.

NeoAxis has a unique game object system that is used to construct the game logic as well as the behavior of the world elements and the way they interact with the player. In NeoAxis everything is a game object whether it is a robot, a crate or a landscape. This is done by a well-defined class hierarchy (see figure 2.11), that allow programmers to create or modify game components according to their needs. This edition is done via C# on Visual Studio.

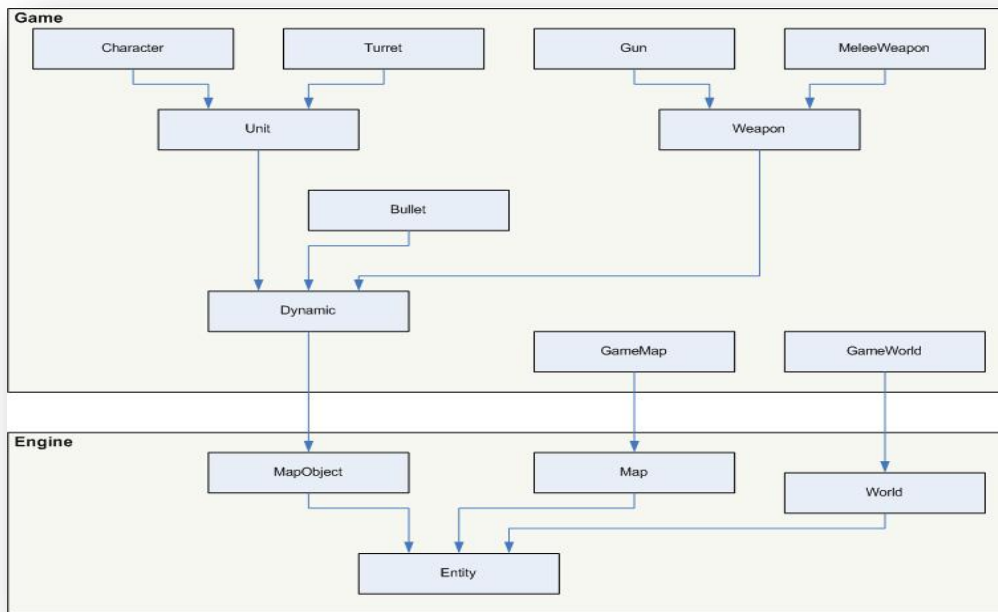


Fig 2. 11 - Game class hierarchy used by NeoAxis Engine.

Engine also has a wide range of pre-made base classes for rapid and intuitive development of games. Some of this classes feature the following capabilities of the engine:

- Water surfaces with physical influence and splashes;
- Cut scene manager that enables the creation of cut scenes in the game;
- Units divided in characters and vehicles. This allows the creation of any unit types, such as characters, tanks, turrets, etc.;
- The ability to create factions for unit groups with some generic IA and path finding algorithms ;
- The Creation of weapons, bullets and physical explosions;
- Several types of doors, including automatic doors;
- A Boolean switch system;
- Start and respawn points. Different start points and helpers for units and other objects;
- Physical streams;
- Camera management system with various pre-programed cameras;

NeoAxis Engine already includes a framework of game types. This framework allows the users to create functional prototypes quickly. Some of the game types included in the framework are: first person shooter with multiplayer support; real-time strategy; and third person shooter.

However, what makes NeoAxis Engine such a good engine for indie developers is the unique set of tools that provides to assist during all the processes of game development.

The Expandable Map Editor (see figure 2.12), is a powerful tool for placing objects and creation of game scenes with full undo/redo support. This editor gives to the developer control over all elements of in-game environment from a graphical viewpoint. In this software is possible to create, place, size, scale and rotate individual game objects, as well as view and modify any of the properties of those objects. The user can easily edit the parameters for static objects, terrain, characters, light sources, water, sky, and other objects in the scene.

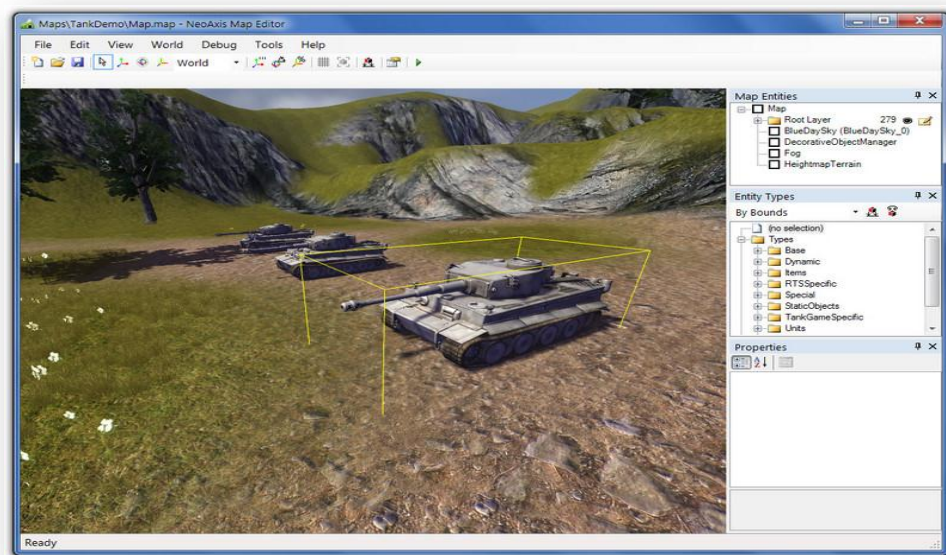


Fig 2. 12 - A screenshot from the NeoAxis Engine map editor tool.

Another tool that the developers will use a lot while working with this engine is the NeoAxis Game Object Editor (see figure 2.13). This editor

allows the configuration of all types of game objects, and to adjust objects visually. It includes appearance creation, configuring physics, particle systems, and all other attributes of game objects visually.

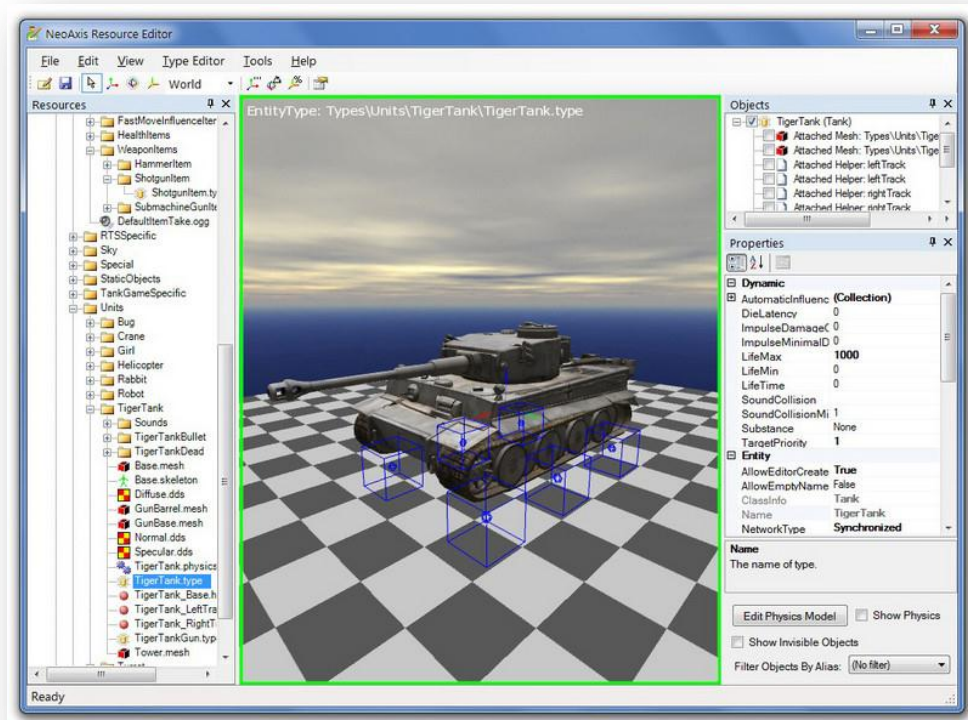


Fig 2. 13 - A screenshot from the NeoAxis Game Object Editor.

These two pieces of software are definitely the main tools of the engine; nonetheless, there are other tools available to assist the developer in the game creation, for example:

- **GUI Editor:** The Graphic User Interface Editor is intended for the creation of end-user controls, menus, dialogues, windows, HUD screens and in-game 3D GUI.
- **Expandable Material Editor:** A visual tool for designing materials and shaders.
- **Physical Model Editor:** Allows the developers to visually configure physical behavior of game objects.
- **Particle System Editor:** A tool designed for the creation of various particle systems.

- **Terrain Editor:** A landscape design tool that supports geometry editing and painting of alpha layers onto terrain to control blending, collision data, and support of detail and normal maps.
- **Static Lighting Calculation Tool:** A static lighting calculation tool with Lightmaps and Irradiance Volume support.
- **NeoAxis Exporter:** An exporter that can be installed on Autodesk 3Ds Max and Autodesk Maya to convert the files to the format used by the engine.

As it was mentioned earlier, NeoAxis Engine is most used by indie developer companies. In the following table we present some of the companies that used this engine and some of the games created by them.

Table 2 - Companies that developed titles using *NeoAxis Engine*.

Independent game development companies	Developed game titles
OHBA [48]	Homura Combat [49] (Third Person Shooter, Arcade Game)
Dream Dale [50]	Elementary My Dear Majesty! [51] (Casual Game, Hidden Object)
Makivision Games [52]	Sacraboar [53] (Real-Time Strategy)
Donsoft Entertainment [54]	Capoeira Legends: Path To Freedom [55] (Action/Beat'em Up Game)
Magrathean Technologies [56]	SickBrick [57] (action-oriented Sci-Fi FPS)
Magrathean Technologies [58]	Incognito [59] (Space Adventure Game)
Clockwork Brains [60]	Plyushkin Syndrome [61] (Arcade Game)

2.4.7. Unity 3D

Unity 3D [30] is other game engine that is an integrated tool for the development of 3D games. Unity's development environment runs on Microsoft Windows and Mac OS X, and the games created by this tool can run on Windows, Mac, Xbox 360, PlayStation 3, Wii, iPad, iPhone, Android, and, in a near future, on Linux. This tool consists on an editor for developing/designing content and in a game engine for executing the game. Unity won the Wall Street Journal 2010 Technology Innovation Award [62] in the software category, and the number of developers using this software is estimated to be around 500,000.

There are two versions of Unity 3D, the standard version, and the Pro version. Unity 3D Pro is more powerful and has more features than the standard one, however, we will focus our attention on the second one, since is free and the most used.

In terms of rendering, Unity 3D takes advantage of a new deferred rendering pipeline, which is a three dimensional shading technique where the result of a shading algorithm is calculated by dividing it into smaller parts that are written to intermediate buffer storage, the G-buffer. This information is combined later, instead of immediately writing the shader result to the color frame buffer. The use of deferred rendering allows the engine to support a big quantity of dynamic lighting.

The engine already has 100 build-in shaders, that ranger from Diffuse and Glossy, to more complex ones, like Self Illuminated Bumped Specular. Unity 3D also uses Surface Shaders, a new simplified way to use author shaders for multiple devices and rendering paths. The engine also uses scalability to make sure that the advanced shader effects will run well across all target hardware.

Unity also has a large collection of full-screen image post processing effects built in, like Sun shafts, high quality depth-of-field, lens effects, curve-driven color correction and others.

Regarding the performance of the engine on rendering, Unity 3D uses a process called batching. Basically, the engine automatically combines geometry into batches, which significantly minimizes driver overhead while retaining full flexibility. It also has LOD support to manage meshes with several levels of detail.

In terms of lighting, Unity 3D presents soft and hard real-time shadows. These shadows can be cast from any light in the scene, and a variety of strategies is used to make them the fastest as possible and even work on old computers. With the linear space lighting and HDR, is possible to create games with stunning looks since the multi-threaded renderer is able to render the scenes with ease.

However, probably one of the best features of this engine is the light mapping (see figure 2.14). This features gives to the developer total control over the game ambience.



Fig 2. 14 - A screenshot from Unity light mapping editor.

Unity light system also supports light probes, in order to give life and realism to light mapped scenes without the high cost of typical dynamic

lights. Dual light mapping is also supported. One light map is used for faraway stuff, while another contains only bounce light. This allows models to integrate nicely with created light maps. Unity 3D is also capable of creating the UV maps on its own, this means that the developer doesn't need to unwrap unless he wants to.

In terms of terrain creation, Unity 3D also provides a set of tools to make this task easier. The unity in-editor tool allows the user to carve, raise, and lower sweeping and mountainous terrains. The engine supports tiled textures, which can be blended and combined with a collection of accurate tools. This allows a handful of low-resolution textures to make diverse types of terrains. The light map features seen previously can also be used in terrains.

Unity's terrain engine is extremely easy to use and allows the user do specific task like painting trees, ground debris, and blades of grass. All of this with configurable distances for foliage rendering transitions to billboards in order to obtain the maximum performance.

The terrain creator works together with the tree generator. This tool allows the developer to use a mix of procedural generation and hand placement freely across different parts of the generated trees. The user can add branches twigs and leaves in real-time, rearranging the tree hierarchies with simple drag and drop operations. Assigning textures to the trees is also an easy task. The user just needs to assign the leaf and branch and twig materials to the tree. Unity engine will automatically atlas all textures together and bake the maps. It also calculates the ambient occlusion and wind factors for the tree. The engine also optimizes the leaf quads from the texture automatically, giving 20% to 30% of fill rate optimization.

Unity 3D uses the powerful physics engine NVIDIA® PhysX® [63]. Basically, we can divide the physic in Unity in five groups:

- **Cloth:** Inside the cloths, we can have two types of simulation, Interactive Cloth that interacts fully with the rest of the

environment; or Skinned Cloth, which is an optimized solution for garments on animated characters.

- **Soft Bodies:** creates soft bodies from objects, for example, semi-deflated objects that interact realistically with the environment.
- **Rigid Bodies:** enable the game objects act under the control of physics. The rigid body can receive forces to make your objects move in a realistic way with no scripting required.
- **Ragdolls:** with the ragdoll wizard it's possible to set up a full ragdoll from animated character in seconds. The user can tweak the ragdoll as needed in order to get unique behaviors.
- **Joints:** Unity processes several types of joints, like hinges, springs, ball-sockets, character limbs, and fully-customizable configurable joint.
- **Cars:** Unity uses a dedicated wheel collider to accurately simulate the traction model of real car tires.

Unity 3D also has some build path finding algorithms to turn the developed games more realistic. The user just needs to define the boundaries of any navigable space, and the engine will do the rest.

The audio on Unity 3D is managed by the FMOD [64] audio engine, using its tool to do the necessary editing and treatment of the audio files.

In terms of scripting, Unity 3D supports three scripting languages: JavaScript, C#, and Boo (a dialect of Python). All three are equally fast and can interoperate, making use of .NET libraries that support databases, regular expressions, XML, networking and so on. The engine game logic runs on Mono [65], an open source .NET platform.

Like it was mentioned before, Unity 3D is an integrated tool, this means that the Unity editor (see figure 2.15), works together with Unity game engine, allowing the editor to do everything a published game can do. The user can instantly run the game inside the editor and preview how it behaves on the several platforms. It's also possible to alter values, assets, and scripts on real time, this give to the developer a change to test

a different gameplay mechanic or just to see how another material might look like in scene.

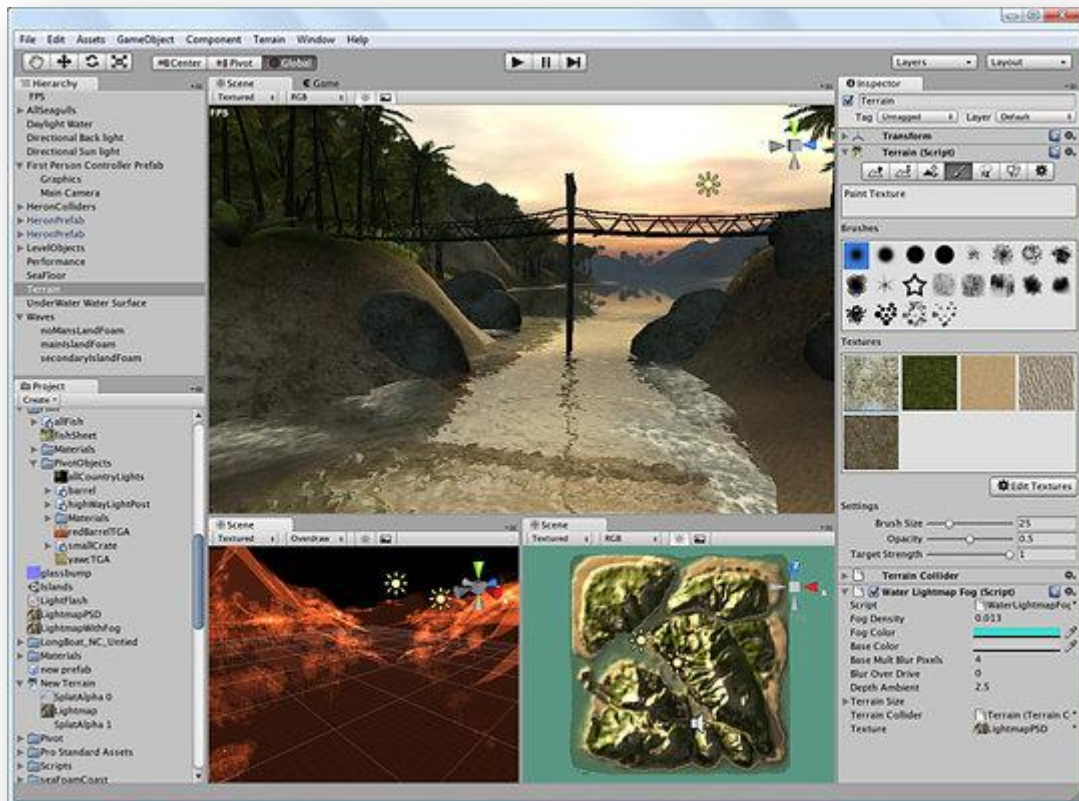


Fig 2. 15 - A screenshot from Unity Editor interface.

To simplify the frequent use of complex objects, Unity 3D allows the use of Prefabs, which are similar to macros. This feature can then be easily placed throughout the game or instantiated at runtime. Any changes done to the original prefab are propagated to all dependents; this means that major and minor adjustments can be made very quickly on a large number of instances.

Unity asset manager helps the developers to organize the assets while working on large projects. The project browser shows everything with interactive previews, tagging and searching.

Unity editor allows the developer to design worlds due to its building tools. These set of tools help the users to assemble assets and set up the

game environments. With operations of drag and drop, it's possible to bring meshes into Unity, and then, with other drag and drop operation, to the 3D world. To preview the game world created, the developer just needs to press the play button. It's also possible to easily add primitive meshes, like spheres, boxes, and capsules to define trigger zones in the game world. Unity helps the developer to visualize these meshes, position, scale and rotate them as he wishes.

The editor is extremely flexible; it enables the user to set-up layouts simply by dragging tabs to the desired position, and a full range of features such as grid (see figure 2.16), and surface snapping allows the user to quickly and accurately position the objects. With the vertex snapping tool objects can be positioned exactly where the developer needs them to be.

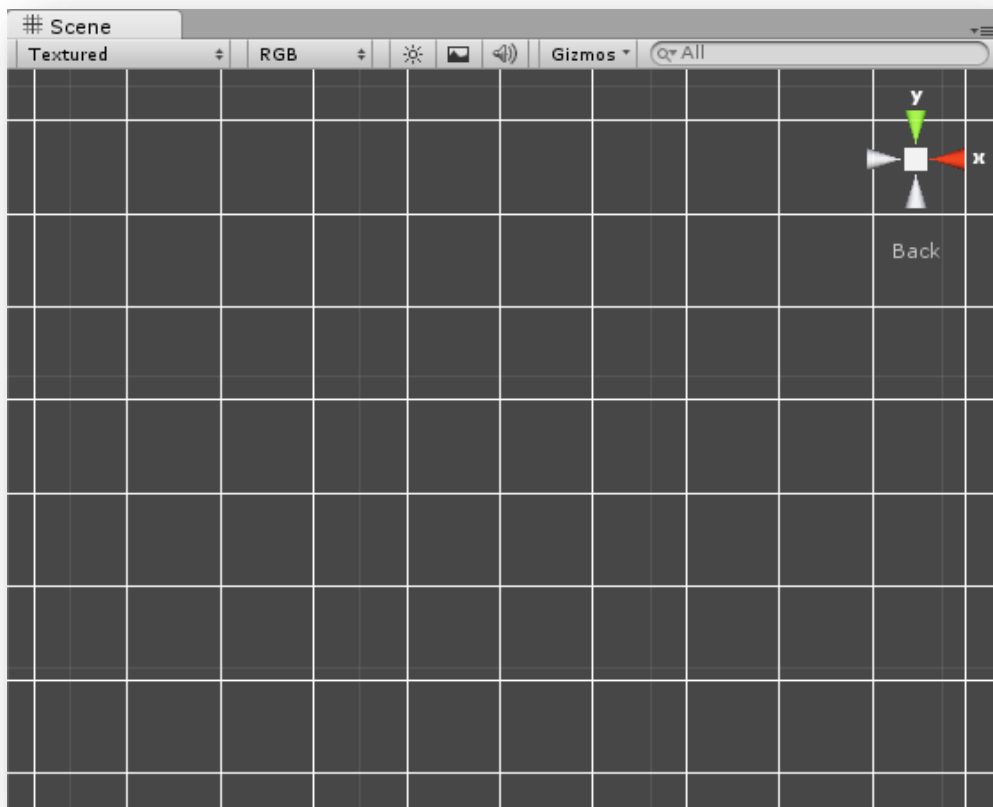


Fig 2. 16 - A screenshot from Unity editor grid feature.

Unity's Asset Pipeline supports all major formats used by CAD applications. All assets in a Unity Project are automatically and immediately imported upon save, whether they are 3D models, animations, textures, scripts, or sounds. The assets can be updated at any moment, even while playing the game inside the editor.

Unity 3D includes a curve and gradient-driven modular particle system tool editor called Shuriken (see figure 2.17). With this editor, the developer can easily adjust individual parameters of each particle system via Shuriken. It's possible to playback a selected particle system, pause it at any point in time to adjust any parameters, and then play it again to instantly see the modifications. It's possible to group individual particle systems into Particle Effects in order to synchronize the systems. Unity also provides a Particle Editor to manage potentially complex Particle Effects. This is accessible via the Inspector View, and allows the user to toggle curves on and off.

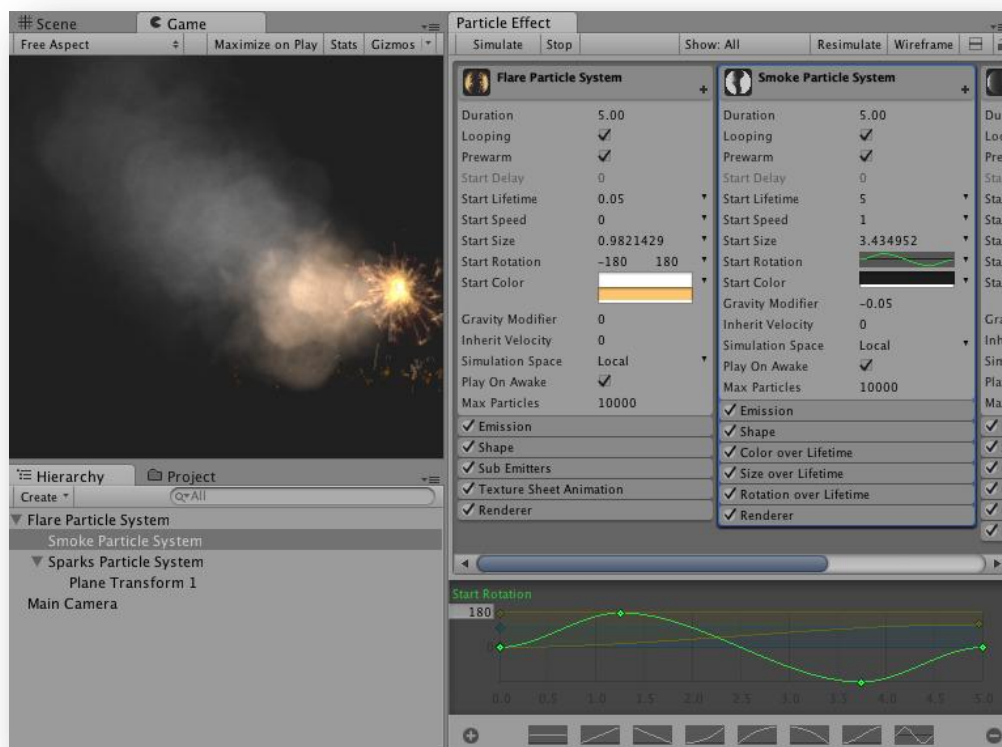


Fig 2. 17 - A screenshot from Shuriken.

A vast list of games has been created using Unity 3D, both for mobile platforms such as iPhone or Android, for portable consoles like PlayStation Vita, and for platforms where we are costumed to see AAA games like Xbox 360, PlayStation 3 or desktop computer. In the following table we present a small list of game companies that used this engine and some of the games they produces using the tools given by Unity 3D.

Table 3 - Companies that developed titles using *Unity 3D Engine*.

Independent game development companies	Developed game titles
GameArt Studio GmbH [66]	A.I. Invasion [67] (MMO)
Crescent Moon Games [68]	Aralon: Sword and Shadow [69] (RPG)
Bigpoint [70]	Battlestar Galactica Online [71] (Adventure)
NPlay [72]	BeGone [73] (Shooter)
SilverTree Media [74]	Cordy [75] (Adventure)
Limbic Entertainment [76]	Dungeon Empires [77] (RPG)
Fun Bits [78]	Escape Plan [79] (Adventure)

2.4.8. UDK

UDK [1] stands for Unreal Development kit, and just like Unity, it's a game-making software, resulting of a collection of development tools, such as editors, scripting systems and compilers. Every game created by this software is powered by Unreal Engine [80], a game engine developed by Epic Games [2]. UDK consist on two fundamental pieces: the Unreal Engine and the development tools (see figure 2.18).

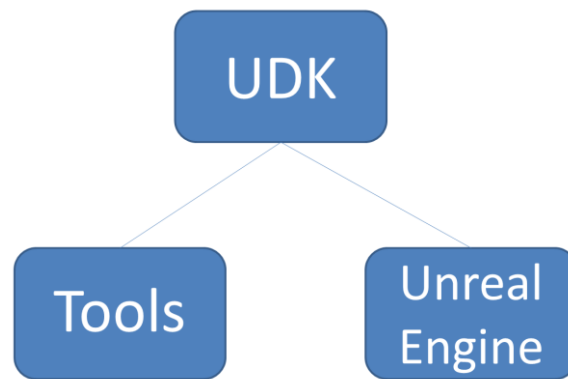


Fig 2. 18 - UDK structure.

The first version of UDK was released in 2009 and marked an important milestone in game development history [81]. For the first time a AAA standard game engine was made available both for hobbyists and independent game developers.

Nowadays exist two different releases of Unreal Engine and its development tools. One is used by engine licensees, given access to the development tools and the engine C++ source code. The second version is UDK, which contains the features of the Unreal Engine and its tools, but no access to the source code. Although UDK does not allow developers to access the engine source code, the engine can be customized through UnrealScript.

UDK differs from Unity in several aspects, but mainly on its workflow. Unlike what happens in Unity, where there is an integrated environment, UDK is composed by several applications that work separately.

UDK Editor (see figure 2.19), is the main editing tool in UDK. It's here that developers spend most of its time. It offers a powerful and flexible GUI that can be controlled by mouse and keyboard. UDK Editor is composed by several sub-editors. Let's take a look at each one of these parts more closely.

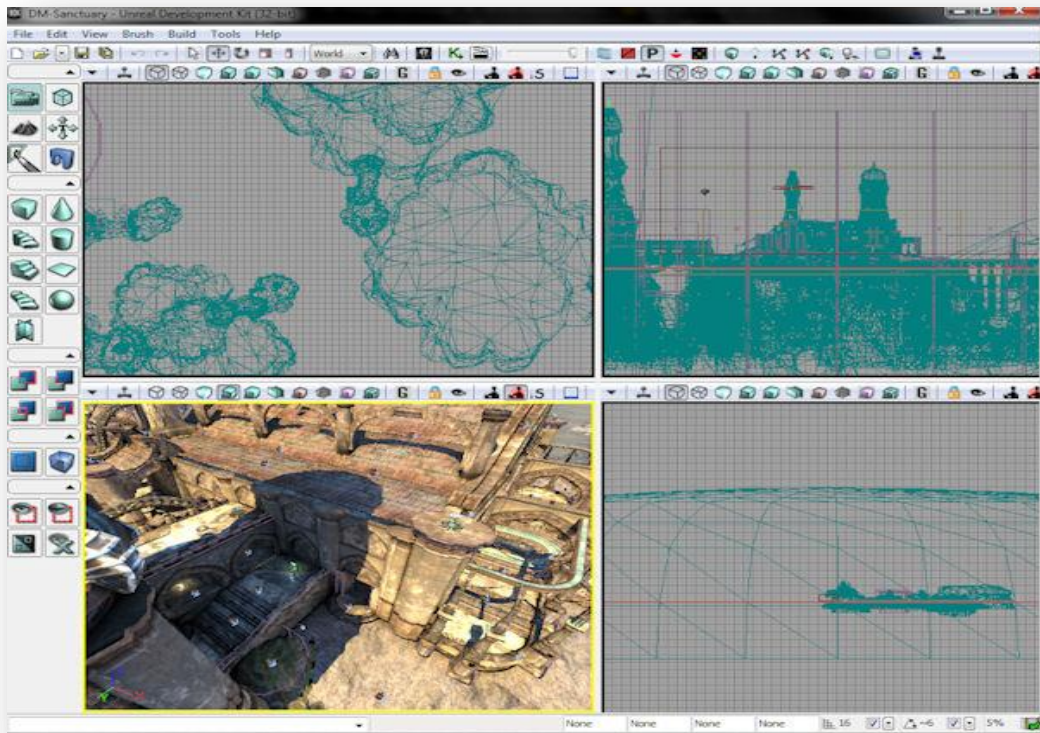


Fig 2. 19 - Screenshot from the UDK editor interface.

The focus of the UDK Editor is the Level Editor. This tool allows the developers to build the game world by drag and drop operations of 3D assets into the 3D environment. Each new element added to the game world is called actor. After importing a new actor to the 3D environment, the developer can select it, position it, orientate it and scale it. The Level Editor also offer a set of tools including the typical Copy, Paste and Delete, as well as organizational tools for naming and fiddling actors, or even grouping actors. The Level Editor also offers real-time previewing of the game levels.

The Matinee Editor (see figure 2.20), is the tool responsible for all animation on the game levels, being the interface of the Unreal Animation System. Here, the developer can create and edit key-frame animations for the actors in the game world. With this tool is possible to create any number of key-frame animations for any number of actors on the level. The

Matinee Editor has several helpful tools for adding, removing, editing and defining key frames, as well as tools for the adjustment of the interpolation curves generated between the frames.

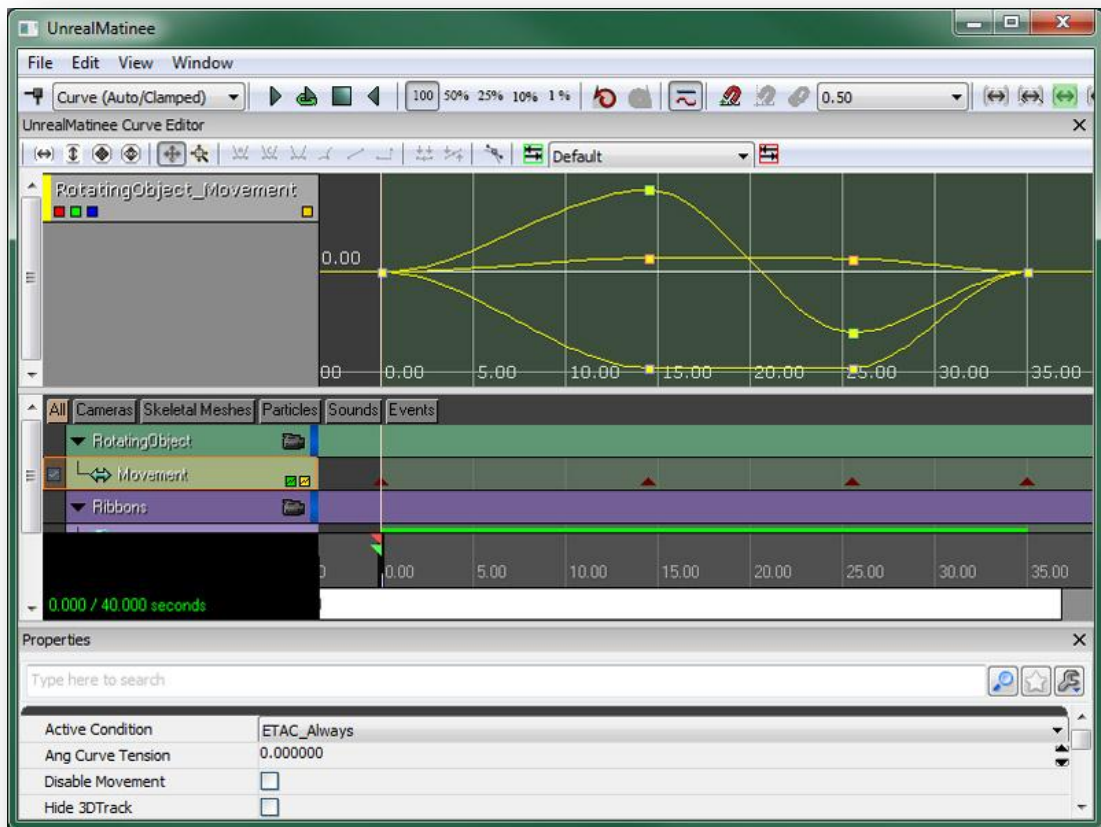


Fig 2. 20 - Screenshot from the UDK Matinee editor interface.

Defining all game logic on the game level can be an arduous task, usually done by programmers. However, UDK Editor has the Kismet Editor (see figure 2.21), in order to simplify this task. Basically, the Kismet is a visual scripting system that uses graphs to define the level logic and flow. The developer uses the mouse to create graph nodes linked by wires which define a certain action of behavior. With Kismet is possible to trigger animations or behaviors, reset levels, play sounds and music, keep track of scores and statistics, and change the lighting, among others...

The Kismet Editor works as a GUI for the UnrealScript language, this means that Kismet automatically generates the code of the defined graphs

for the level. It's also possible to customize Kismet using UnrealScript, this way the developers can extend the behaviors already defined in the editor adding their own custom behaviors as visual nodes.

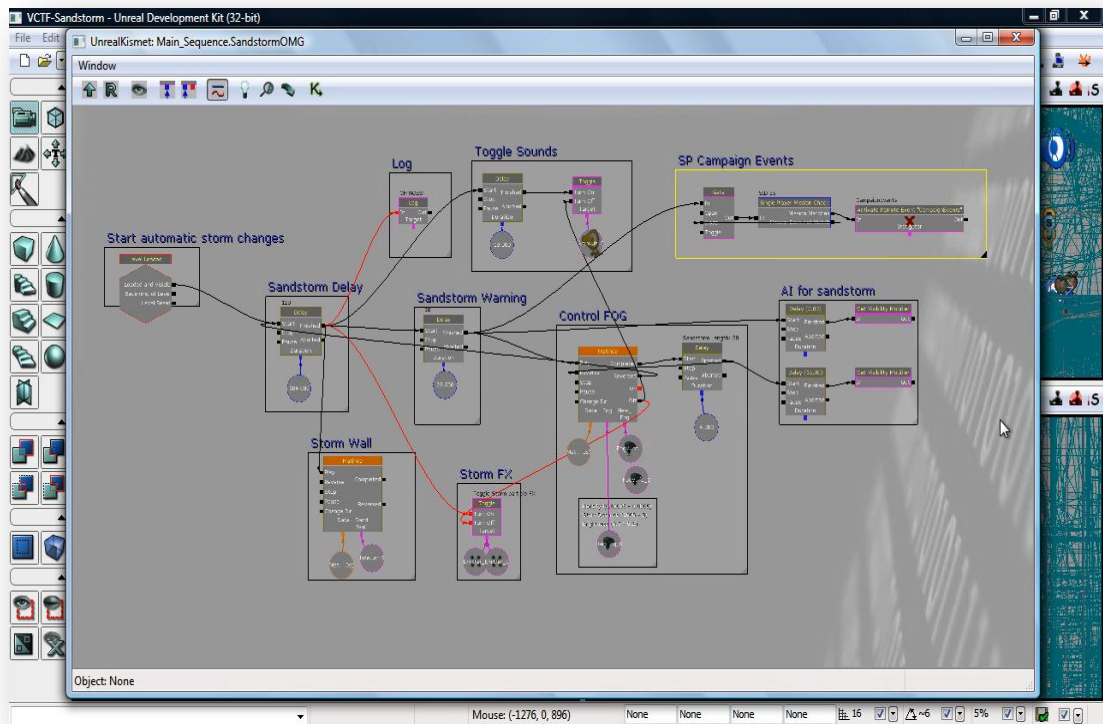


Fig 2. 21 - Screenshot from the UDK Kismet interface.

The UDK Material Editor (see figure 2.22), allows developers a way to import, create and edit materials that are applied to the actors in the level, and all this done by mouse-drive operation. This editor not only lets the user choose which image should be applied on an actor, but also the surface type of the 3D models, offering a vast list of options like shiny, reflective, chrome-like, rough, smooth, or soft, among others...

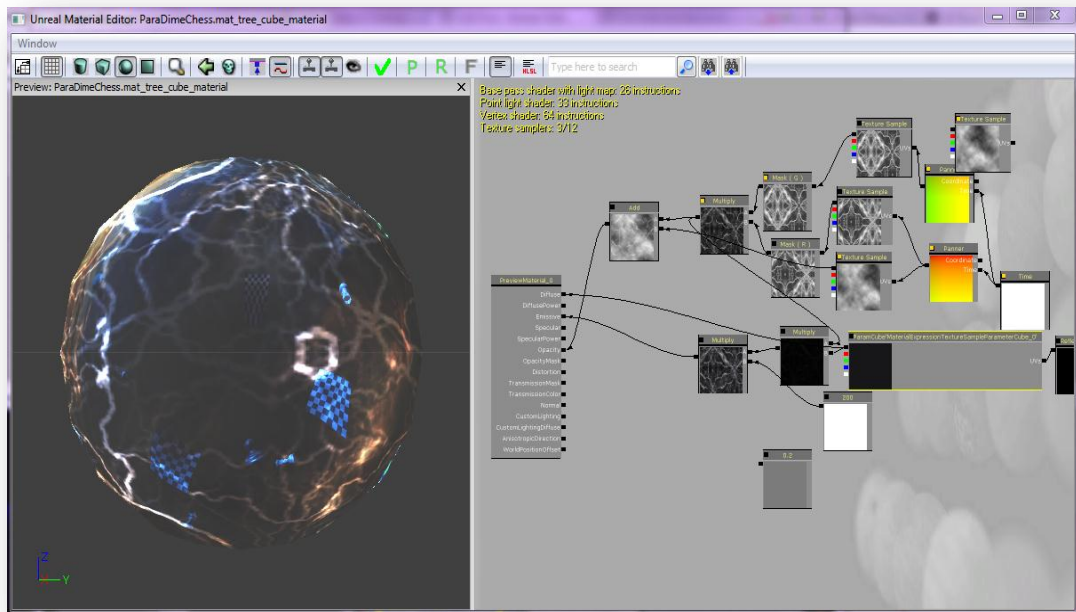


Fig 2. 22 - Screenshot from the UDK Material editor interface.

The Sound Editor (see figure 2.23), as a GUI that allow the users to import WAV files into the engine and lets them apply effects and processes over the imported files, for example echo effects, mixing effects, loop instructions or positional effects. The result of the effects and processes is the one that is going to be added on the level editor as an actor.

UDK offers for sounds the same type of operations that it offers to other actors, like scaling, positioning and rotating. This is very important since the way the sound is heard by the gamer is influenced by its position on the level. Also, the falloff of any sound can be controlled by the developers; this is done by surrounding every sound in the game by an invisible sphere, the size of the sphere defines the sound volume, from full sound to mute, where the full sound is at the center of the sphere.

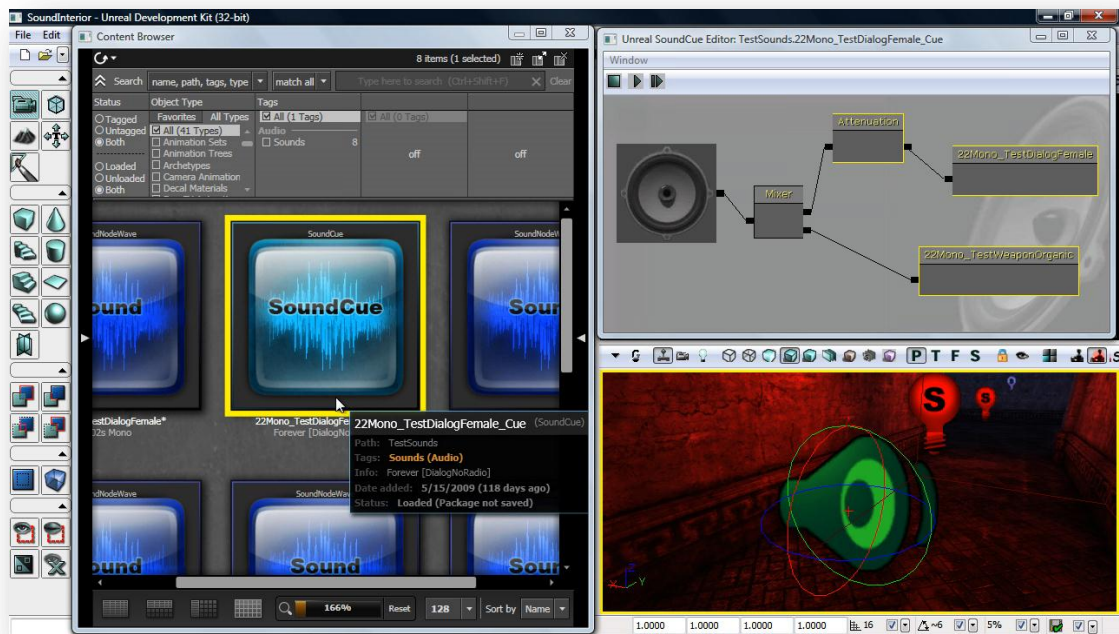


Fig 2. 23 - Screenshot from the UDK Sound editor interface.

The Cascade Editor (see figure 2.24), is the one responsible for managing the particle systems in UDK. Basically, it offers a visual paradigm to the construction of particle systems, allowing the user to test the created systems in real time. The created systems are divided into two pieces: the emitter, and the particle. The emitter is one responsible for spawning or emitting the new particles into the game level, therefore, it has a specific location on the level and associated logic to define how it will behave. The particle is a descriptive object that the emitter uses as a template to produce new particles.

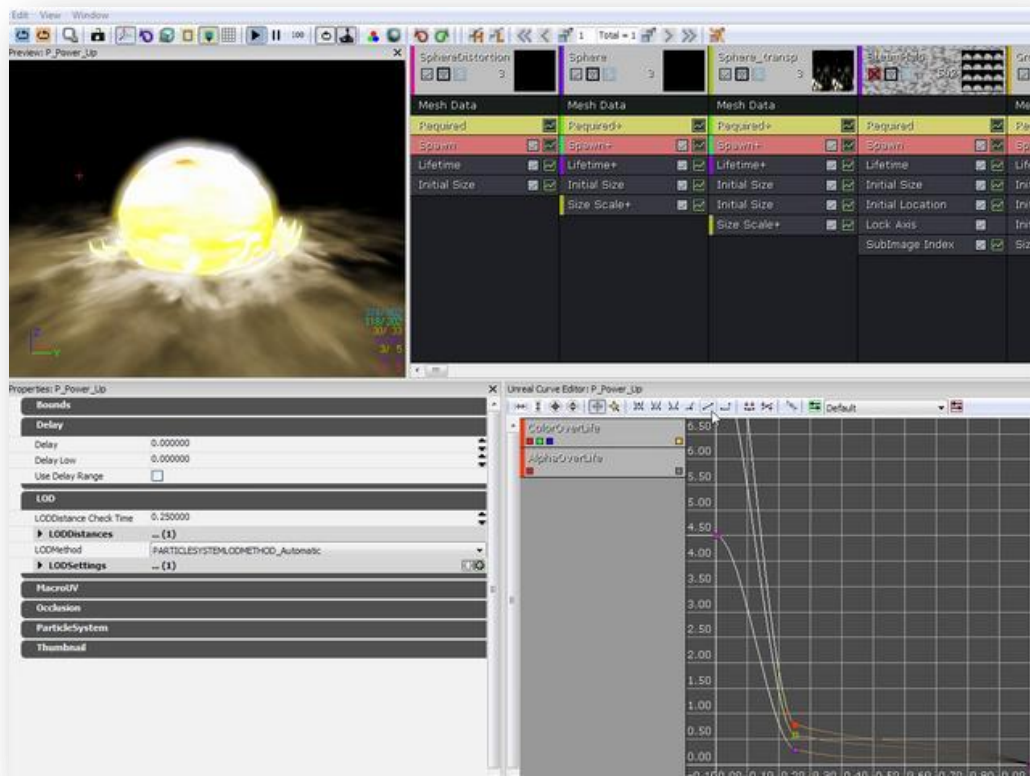


Fig 2. 24 - Screenshot from the UDK Cascade editor interface.

Besides the UDK Editor that we have been seeing until now, there are other tools on UDK very helpful for the developer and with very specific purpose.

Lightmass (see figure 2.25), is the lighting system supported by UDK. This tool allows developers to use a range of lighting source types that vary in the way they cast light, for example, point lights, directional lights and spotlights. The lights can then be placed on the level using the Level Editor, creating actor from the lights. This means, that just like any other actor, their properties can be altered, allowing the developers to define how the lights is casted and how it behaves.

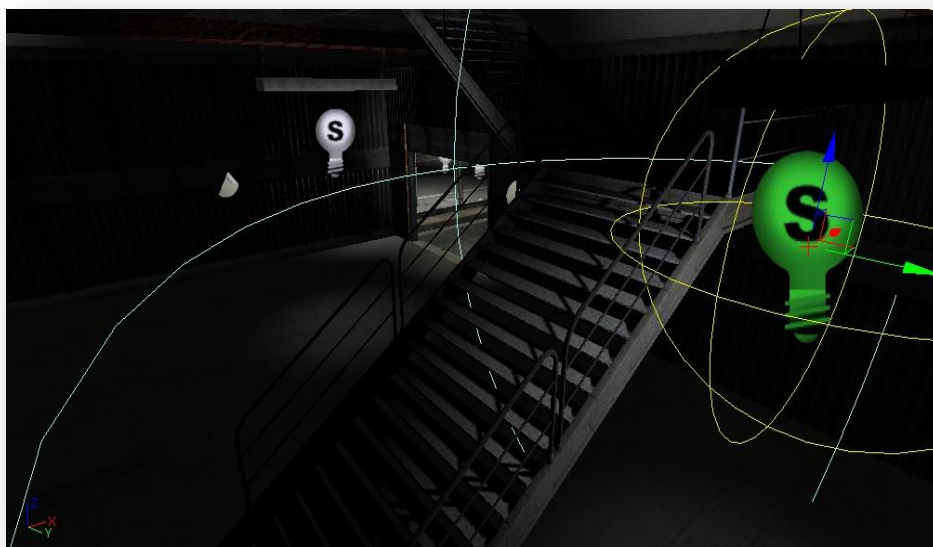


Fig 2. 25 - Screenshot from the UDK Lightmass editor interface.

The UDK Terrain tools (see figure 2.26), allow the developers to construct a rough landscape that will work as the base that defines the general extend and contours of the level. Using this tool the developer can create mountains, rivers, forests, and so on. After the terrain is created, the user can then add more detail by adding meshes to the level. When the developer starts to build a new level, he begins with flat plain of geometry, that he will then edit using a set of terrain brush tools to sculpt the level. The available brushed of terrain edition are similar in most ways to those existing in photo-editing software, enabling the user to control the shape, size, softness or hardness of the brush. This tool also allows the developer to import heightmaps that can be turn into meshes.

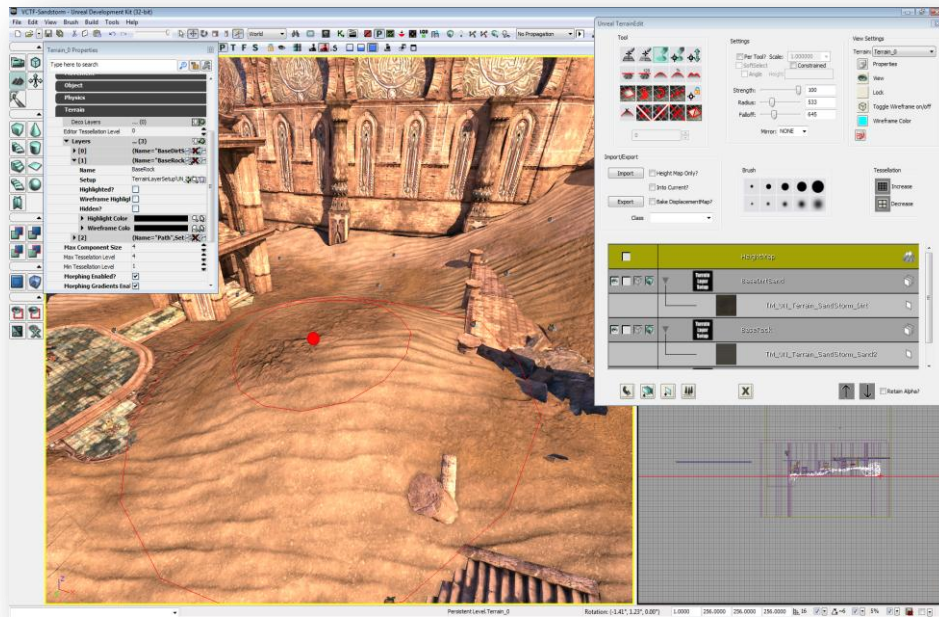


Fig 2. 26 - Screenshot from the UDK Terrain tools interface.

Sometimes it's necessary to adjust the camera during the game play, more precisely during the cut scenes that may take place on the game. For this task, UDK has a set of Cinematic tools (see figure 2.27) that allow the developers to animate almost all the properties of a camera, including position, orientation, focal length, zoom, depth of field, an others.

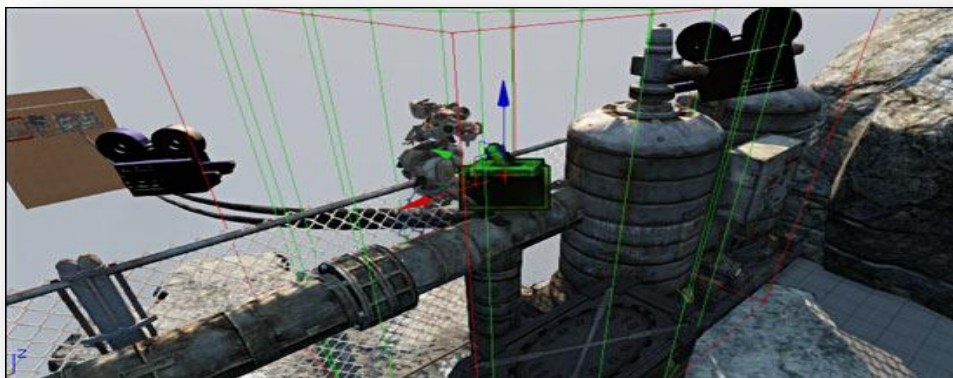


Fig 2. 27 - Screenshot from the UDK Cinematic tools interface.

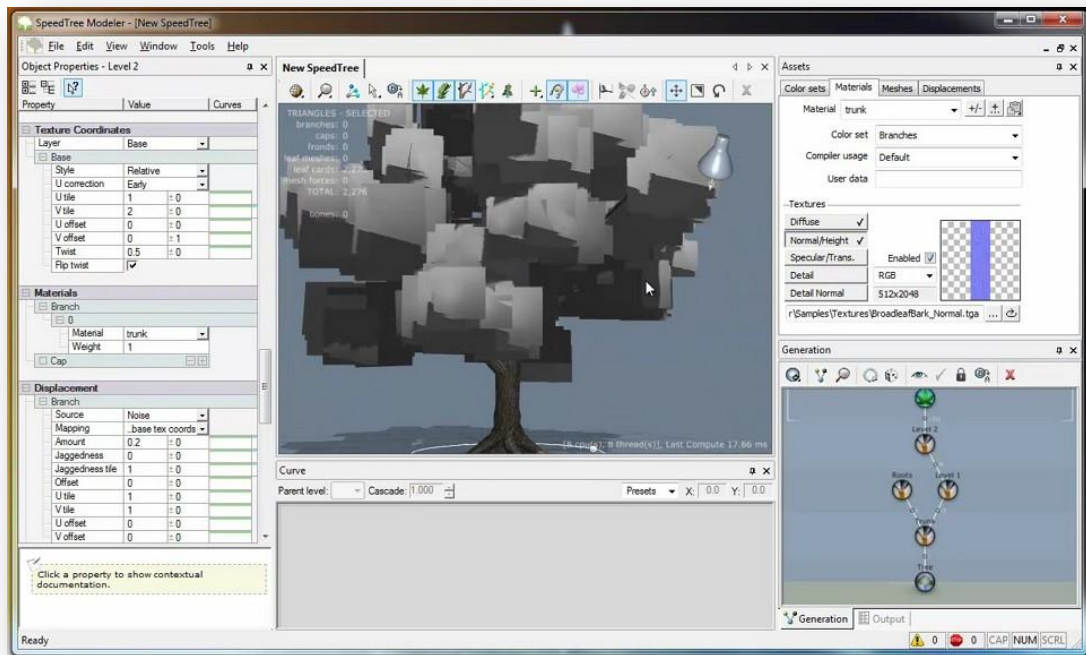


Fig 2. 29 - Screenshot from the UDK SpeedTree editor interface.

FaceFX (see figure 2.30), is a tool in UDK designed to give the developers control over lip-sync and expression control on game characters. One can define lip-sync as the synchronization and animation of the character lips with the vocal track is played for his or her voice when is talking. We define expression control as the ability to manipulate the expressions on character's face to convey emotion, for example, raising the eyebrows, smiling, blinking, laughing, and crying, among others. FaceFX offers several features that help the user to control the animation of character faces.

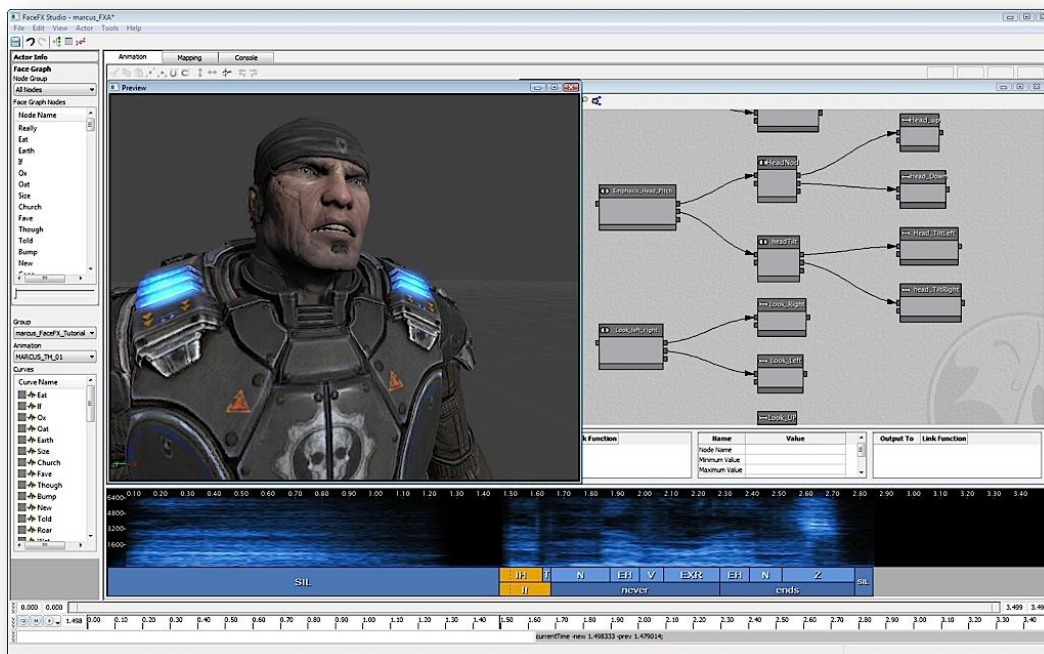


Fig 2. 30 - Screenshot from the UDK FaceFX editor interface.

As it was previously seen on this section, the programmers can use the Kismet to define logic and behaviors for the game; nonetheless, Kismet has its own limits, and sometimes the programmer must use scripting language to define one particular behavior. UnrealScript is the scripting language used by Unreal Engine, and it's based on languages such as C++ and Java, being class-based and object-oriented.

UDK does not provide a native editor for scripting, instead, the programmers can use standard text files, or use Microsoft Visual Studio in combination with the nFringe plug-in for both IntelliSense and code-completion features (see figure 2.31).

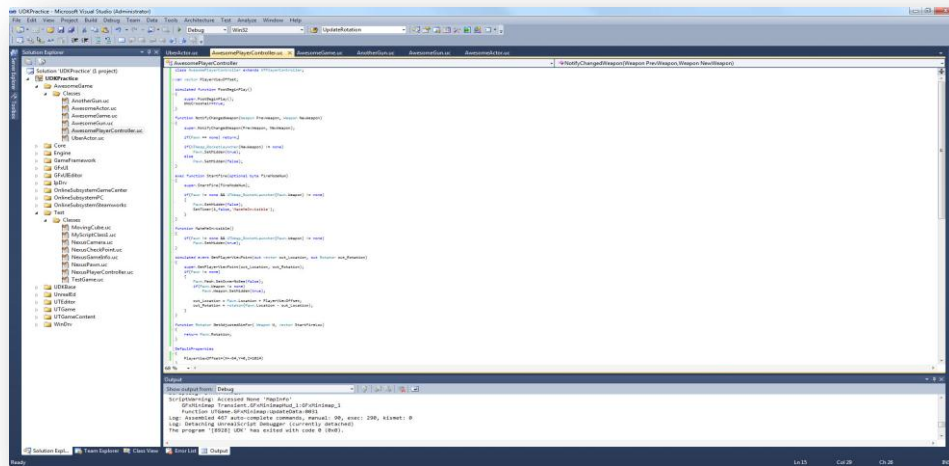


Fig 2. 31 - Screenshot from the Microsoft Visual Studio editing UnrealScript.

On the following table we present some games that have been produced with UDK and Unreal Engine.

Table 4 - Companies that developed titles using *Unreal Engine*.

Independent game development companies	Developed game titles
High Moon Studios [82]	Transformers: Fall of Cybertron [83] (shooter)
Rocksteady Studios [84]	Batman: Arkham City [85] (Adventure)
BioWare [86]	Mass Effect 3 [87] (RPG)
Epic Games [2]	Gears of War Series [88] (Shooter)
Grasshopper Manufacture [89]	Shadows of the Damned [90] (Adventure)
inXile Entertainment [91]	Hunted: The Demon's Forge [92] (RPG)
2K Games [93]	BioShock 2 [94] (Adventure)

2.5. Summary

This chapter reviewed the state of the art concerning game development and game makers and presented them in general taking into account its most relevant aspects.

After a brief introduction, Section 2.1 described a summarized history of games, from the first games to the digital era. In Section 2.2 were presented the most relevant aspects concerning game development in the industry nowadays. The Section 2.3 revealed importance of game engines and frameworks, and how they help in the development process of a game.

Finally, Section 2.4 overviewed the most relevant game engines use nowadays, and its features.

3. Clean World - The 3D Game

3.1. Introduction

Clean World is a serious game that approaches the importance of recycling and renewable energies for the preservation of planet Earth. For the development of this game, it was used XNA 4.0 with the SunBurn [95] render engine, and DigitalRune [96] physics engine. The choice to use XNA was done because the developed game was used to participate in Imagine Cup 2012, in the Xbox360 game design category, in which ended up classified on 7th place on the worldwide finals. According to the rules of Imagine Cup 2012, all games presented on the completion had to be built on this Microsoft technology.

Initially, it was started the development of a tool for the creation of terrains, however, due to the time constraints imposed by the competition, the work focuses primarily on the development of the game, leaving the creation tools to a second phase, where the tools would be created with the purpose of optimizing the creation of content for the game.

For the creation of this game, it was formed a team composed by four elements in which I took part. The following table presents the members of the team and the developed work by each one.

Table 5 - Clean World development team.

Team Member	Developed Work
David Casteleira	2D concept art
Pedro Pereira	Screenwriting
	3D modeling and animation
	Level design
	2D GUI of the game
André Barbosa	Game core programmer
João Dias	Mini games programmer

Clean World (see figure 3.1), is a 3D game created with the purpose of producing awareness about environmental problems that we face today, and tries to show what we can do to protect nature and keep our planet clean from pollution. The game combines classical platform and RPG elements, such as quests. The player will receive several quests related with environmental problems that he has to complete, in order to progress in the game and storyline. These quests may range from picking up a specific item, completing a puzzle or completing a mini-game based on the theme of the game.



Fig 3. 1 - Screenshot from the game Clean World.

During the game, the player will then travel through the *Cypricene* Island, to clean the landscape, recycle objects and convert factories and machines to use clean energy. All this work will be done by a small robot controlled by the player, which has to use several skills that will be acquired during the game. These skills include transforming into a sphere to roll, absorbing solar energy to recharge, collecting garbage to recycle, among others that will be obtained as the player progresses in the game.

Throughout the game, the player's actions of creating awareness to environmental problems are reflected in the game's characters that soon realize that something has to be done to protect our sacred environment. And so, by progressing within the game, factories will start using environment-friendly energy forms, such as solar panels and wind energy, which results in a cleaner environment and also contributes to Kate's healing process and preservation of the planet Earth.

To make the game more engaging and fun to the player, several mini-games and puzzles will be spread along the island. These mini-games and puzzles are all aimed to create awareness and teach some basic educational content about the environment and how to protect it, such as garbage separation. But they also introduce new gameplay concepts, in order to provide a diverse experience. This way, we hope that the game can teach important concepts about the protection of the environment, through a fun and engaging experience.

In the following sections of this chapter, is going to be presented the developed work done by me to the final version of the developed game.

3.2. Game Storyline

The game takes place on the *Anglas* islands, in the year of 2022. Due to the greed of big corporations, planet Earth is now completely polluted. People can't walk on the streets without breathing masks due to the

polluted air, and the big cities became giant industrial complexes that try to explore to the maximum the last resources of a dying planet.

In one of the *Anglas* islands, the remote island of *Cypricene*, Kate a 16-year-old girl struggles against a disease that now affects almost the entire human population. Kate is alone on the island and she's too weak to get out, so she uses technology to find help. She builds Boris, a small robot with unique abilities, which she sends in search of help.

Boris leaves in search of the medical center on the island. Once he reaches his destiny, he meets Dr. Jacob, brilliant scientist and doctor. The doctor explains to Boris that Kate is sick because she was infected with the Stigma, a new disease that has been affecting the human population almost for a decade.

Dr. Jacob explains that the Stigma has no known cure, however, he was been studying the disease for years, and he believes that the illness is connected to the pollution in the world. He describes to Boris his theory, telling to the robot that if the world gets cleaner and free from pollution, the disease will probably disappear.

With the medicine on its hand, Boris goes back home to give Kate her medication. Feeling a lot better, Kate is willing to test Dr. Jacob theory with Boris help. Together, they start to clean the island from pollution by collecting garbage and stopping machines.

Later on, Boris meets Tom A. Toe, the engineer responsible for the Wastewater treatment plant of the island. With the help of Tom, Boris starts to convert the plant and the machines to use clean energy sources, like the sun or the wind.

Once Boris finishes his job on the island, recovering the environment, he and the other characters leave on a ship to the other islands to help people with Stigma and to clean the rest of the world.

3.3. Modeling and Animation of the Characters

In this section are presented the developed characters for the game, from their background, to the modeling, texturing and animation. During the process of modeling the characters, it was necessary to work together with a concept artist that had the job of created the model sheets. We will start by presenting the main character of the game, which the player controls, and then the other characters that the player will find during his adventure. All game characters and animations were created using Autodesk 3Ds Max 2012 [97] and Adobe Photoshop CS5 [98].

3.3.1. Boris

Boris (see figure 3.2), is a small droid built by Kate. This loyal droid has the mission of finding help to save Kate, who is very sick and weak. Boris is equipped with unique abilities that allow him to overcome the most difficult challenges.

Boris as several abilities that the player can use during the game, most of these powers are based on upgrades that the droid unlocks. Each new upgrade to the droid implies a new model, with new animations. Let's take a quick look at Boris models and animations.

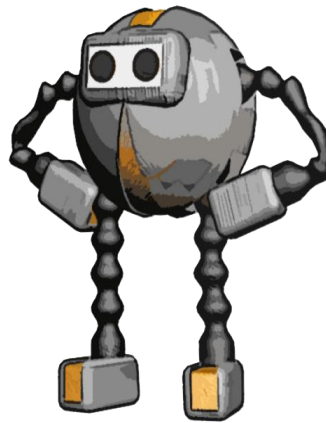


Fig 3. 2 - Boris, the hero from the game.

The first model created was Boris on his robot form. In order to model the character, the concept artist of the team created the respective model sheet (see figure 3.3).

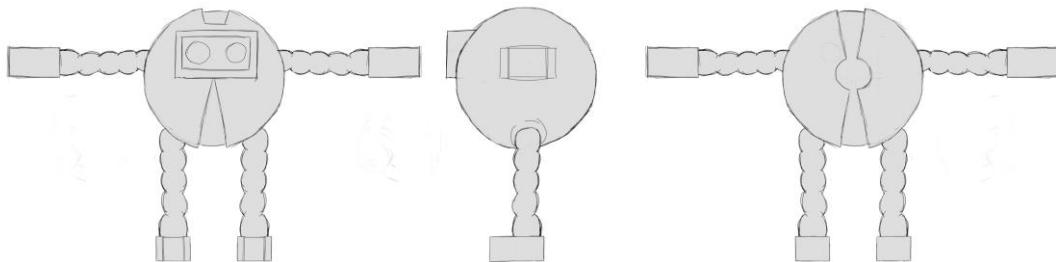


Fig 3. 3 - Model sheet of Boris in robot mode.

Based on this model sheet, was created the main character of the game (see figure 3.4). With the 3D mesh created, it's now necessary to create the textures for the model in order to have the desired look on the game. To create the textures, was used the technique called UVW unwrap. Basically, this technique consists on creating a plant of the 3D model into a 2D image, which will be later used as a map to create the textures.

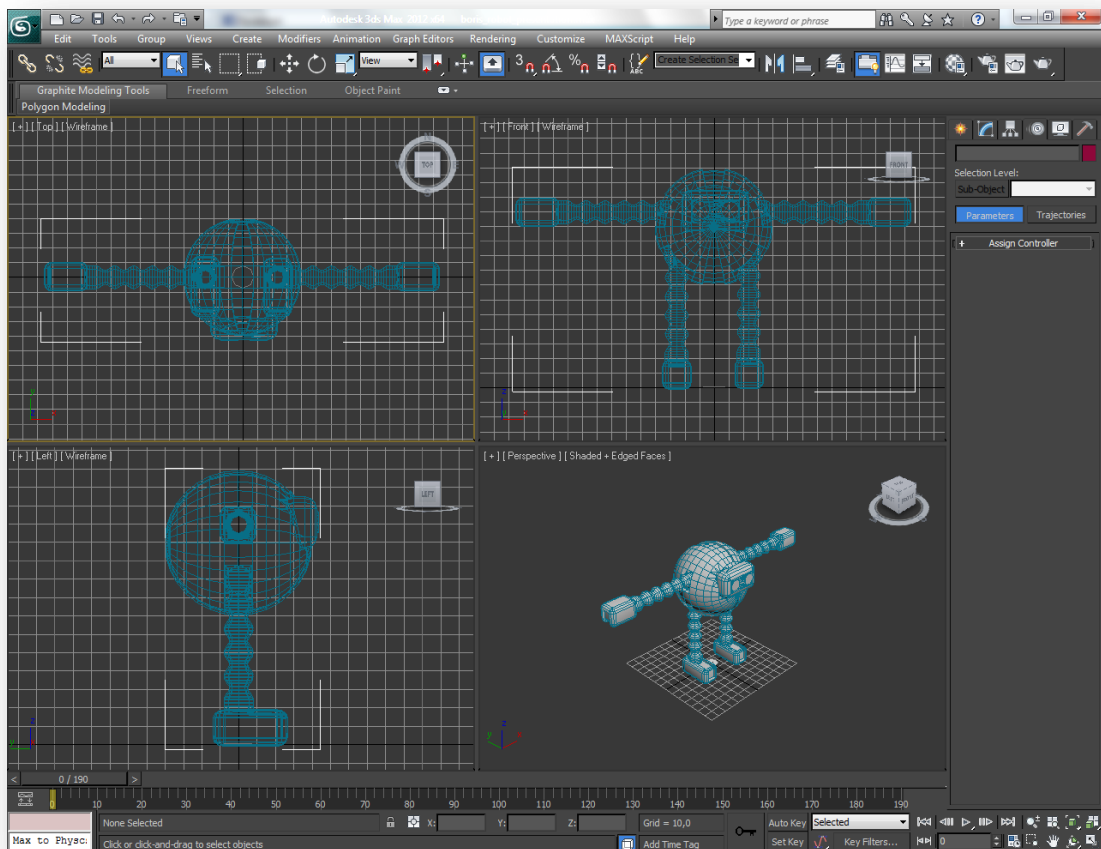


Fig 3. 4 - Model of Boris in robot mode in Autodesk 3Ds Max 2012.

3Ds Max already contains a build-in modifier to perform UVW unwrap, which allows to save the UVW map into uvw files, where it keeps the coordinates of the pieces that compound the model, and also enables the user to render a 2D image (see figure 3.5) to create the textures on 3rd party software.

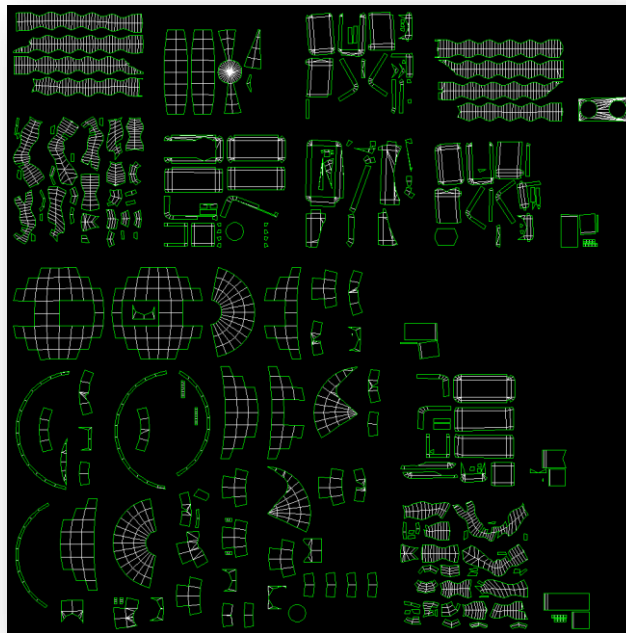


Fig 3. 5 - UVW map of the model of Boris in robot mode.

With the UVW unwrap image is now possible to create the textures using Photoshop CS5. According with the indications on the sheet, the textures were painted over the 2D image (see figure 3.6).

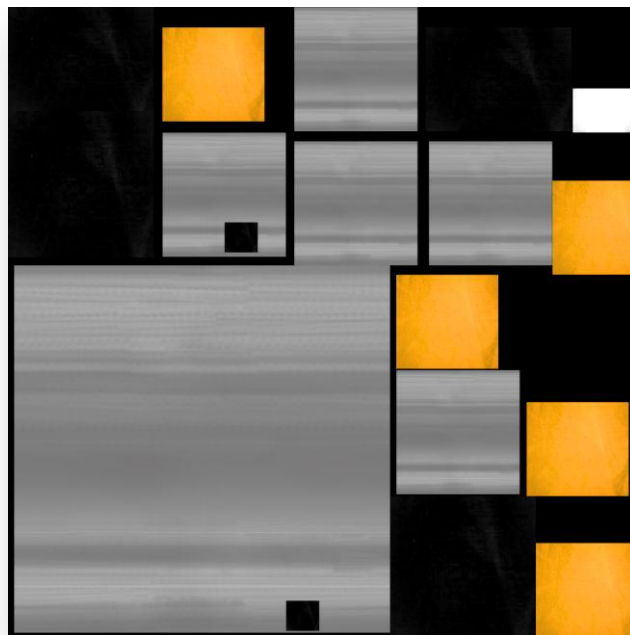


Fig 3. 6 - UVW map of the model of Boris in robot mode with textures.

Once the UVW map was finish, the image file was load to 3Ds Max, and applied to the 3D model as a texture (see figure 3.7).

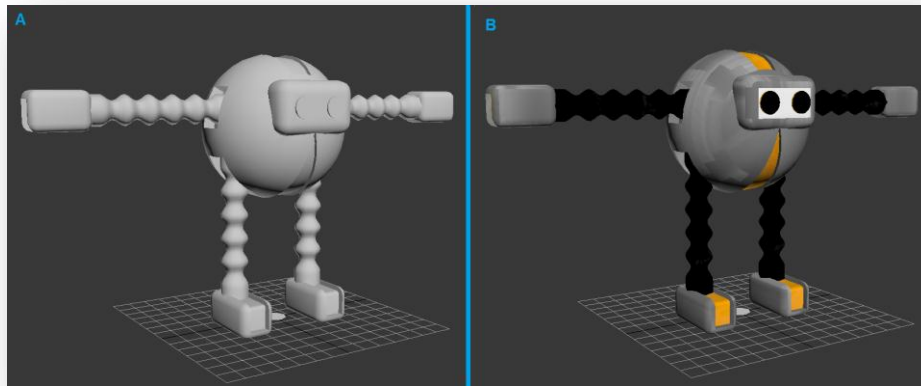


Fig 3. 7 - A: Model of Boris without textures; B: Model of Boris with textures.

With the model created and texturized, the next task is to create the animations that are going to play inside the game. To make the 3D mesh move, it's necessary to create a bone structure attached to it. To do this, it was used the 3Ds max biped, which is a complete body skeleton that can be customized to fit in any 3D model. Once the biped was inserted on the 3D scene, is necessary to make it fit into the 3D Character (see figure 3.8).

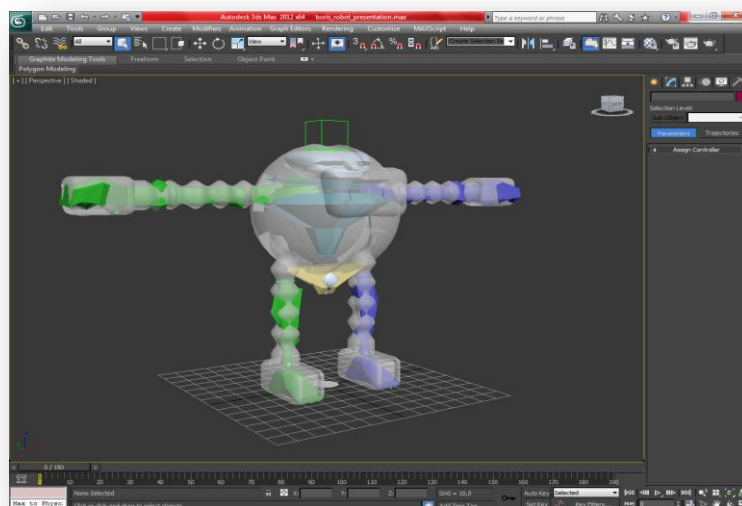


Fig 3. 8 - Model of Boris with biped inserted.

With the biped in place, was used the Skin modifier to attach the bones to the 3D mesh. Now, every time we move a bone of the biped, the mesh moves too. However, these moves are not perfect leaving some 3D mesh behind or deforming the 3D mesh in strange ways. This means that is necessary to adjust the envelopes of the bones (i.e., attach the correct vertexes of the mesh to the desired bone). The envelopes work as a border to define area of influence of each bone (see figure 3.9). If those areas are not well defined, the mesh will behave in strange ways when moving.

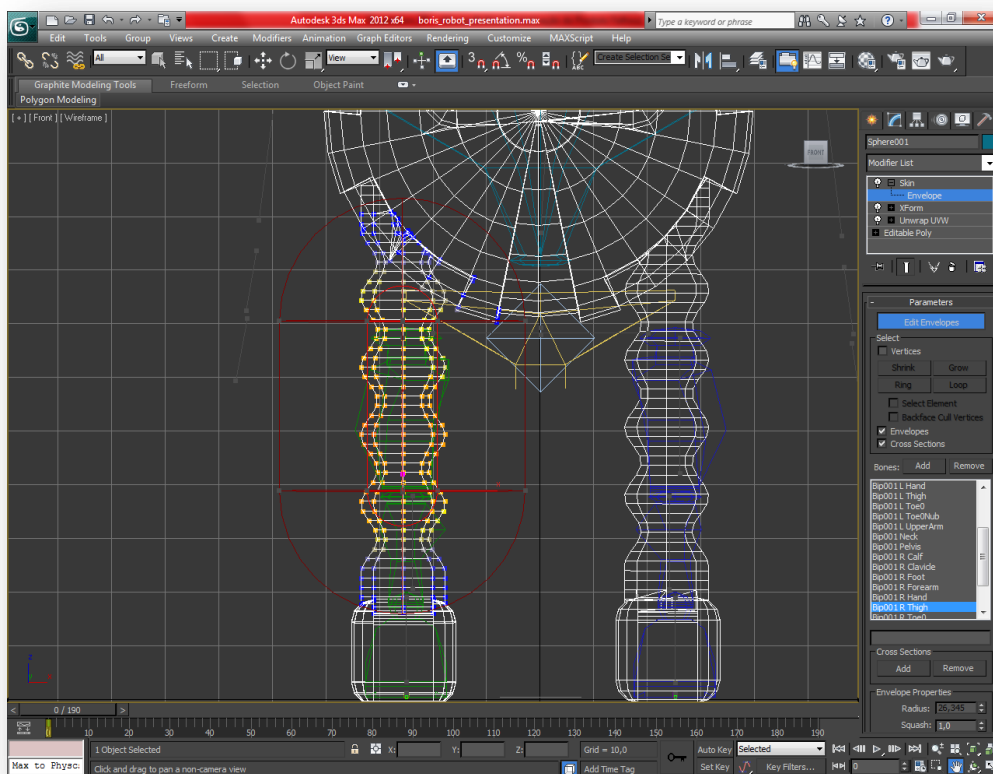


Fig 3. 9 - Adjusting the bones envelopes to the 3D model of Boris.

Once the envelopes are in place, it's time to start creating the animations for the character. Since Boris is controlled by the player, that means that is necessary to animate every action that the player can do while playing. 3Ds Max uses key-frame animations, this means that the user arranges the position, orientation and scale of objects in two time frames, and then 3Ds Max automatically generates the frames in between.

For this model of Boris in were created the following animations:

- Idle;
- Run;
- Run backwards;
- Jump;
- Push;
- Grab on edges;
- Move on edges to the right;
- Move on edges to the left;
- Lift objects;
- Grab a ladder;
- Climb ladders (up);
- Climb ladders (down);
- Transform into sphere;
- Transform into robot;

With the animations done, it's now time to export the 3D model to FBX, and then to X. This must be done because XNA only supports models in FBX or X. The conversion from FBX to X is not necessary if we are talking about static meshes, however, since we are working on a character, this must be done for two reasons: first, XNA does not support biped correctly, which means that the animations will not play the desired way on the game. In order to solve this issue, one must export the model as FBX and imported to Max once again, so 3Ds Max can convert the biped to a regular bone structure; secondly, although this may solve the problem of playing the animations correctly, Max FBX exporter does not allow the user to define animations by frames. For example, if the user wants to name the animation "walk" from frame 0 to frame 60 and animation "jump" from frame 60 to frame 100, it's not possible. The FBX exporter combines all animations into a single animation, and that means that the programmer can't choose an animation by its name to play in game. To solve this, the FBX models were exported with Panda DirectX Exporter [99] plugin for 3Ds

Max, which allows the creation of borders on the animation timeline, allowing the user to define several animations with different names.

Once the first mode of Boris is finish and running on the game, it's time to create the other modes of Boris. After unlocking the first upgrade in the game, the player can turn Boris into a sphere, which means that is necessary to create another model of Boris, this time in sphere mode.

Since the robot mode was already created, we used it as base for the creation of the sphere mode, erasing the arms and legs (see figure 3.10).

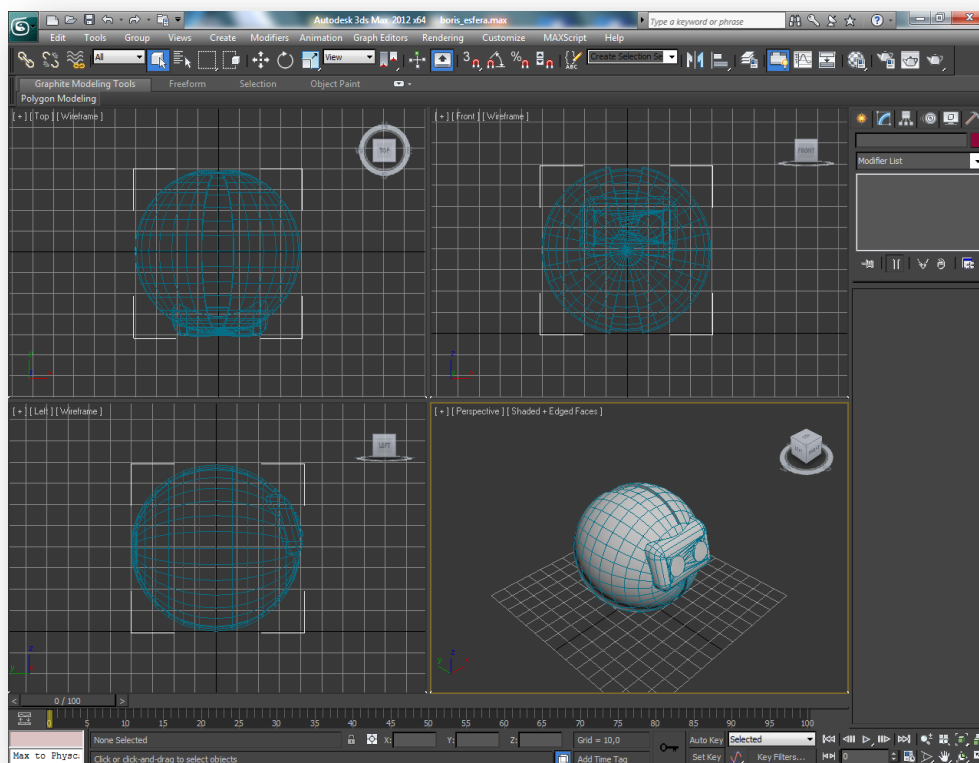


Fig 3. 10 - Model of Boris in sphere mode in Autodesk 3Ds Max 2012.

Just like with the robot mode, it's now necessary to create the textures for the sphere mode model. To do that, we used UVW unwrap once more, and created a map of the faces of the model on a 2D image (see figure 3.11). With the image created, the textures are produced on Photoshop.

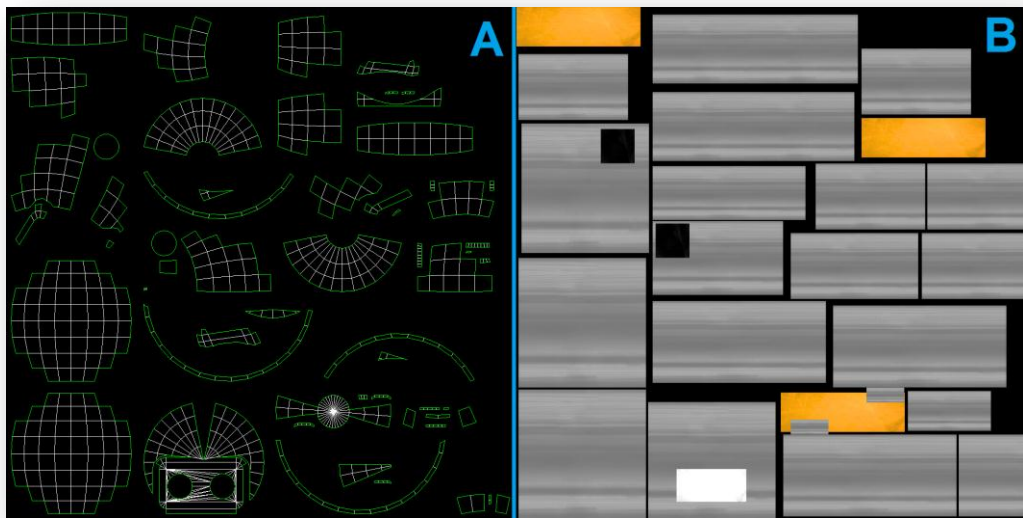


Fig 3. 11 - A: UVW map of the model of Boris in sphere mode; B: UVW map of the model of Boris in sphere mode with textures.

With the UVW map was finish, the image file was load to 3Ds Max, and applied to the 3D model as a texture (as shown in figure 3.12).

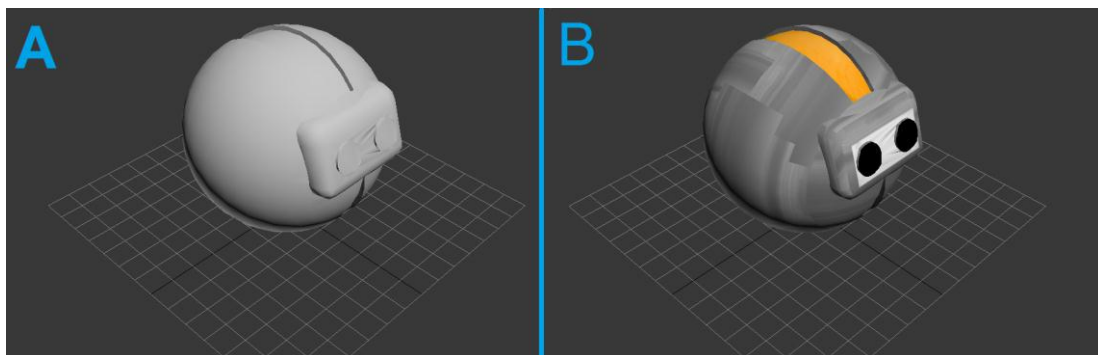


Fig 3. 12 - A: Model of Boris without textures; B: Model of Boris with textures.

Since Boris in sphere mode has no limbs, there is no justification for the use of such complex bone system as the biped. However, since we are going to use the model on the game and use animations, we still need to have one bone attached to the mesh. The model is quite simple in terms of animation, the only thing it does is spin, so a single bone will fix take care of that situation (see figure 3.13).

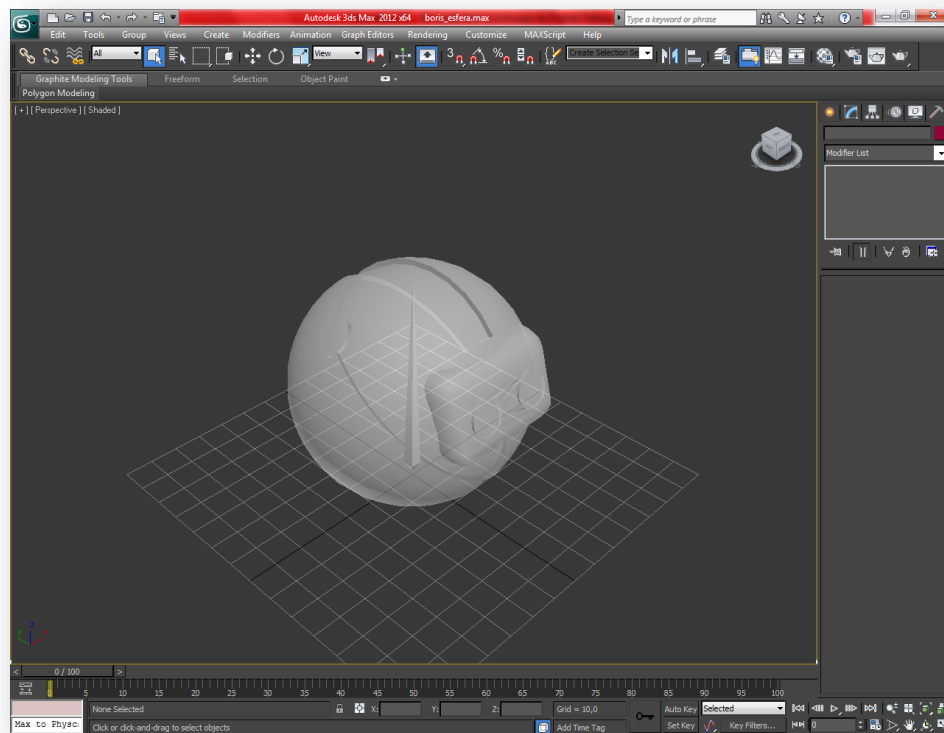


Fig 3. 13 - Model of Boris in sphere with bone inserted.

Just like before, is applied the skin modifier to connect the bone to the mesh. Adjusting the envelope in this situation is quite easy, since the envelope must include all vertex of the mesh. After this, the model is exported to FBX and then to X, by the same reasons mentioned before.

Boris still has a third mode, where he has attached to his back a set of solar panels that he can use to recharge his batteries. Once again, to model this version of Boris, the robot mode was used as base, and then added the panels on its back (see figure 3.14).

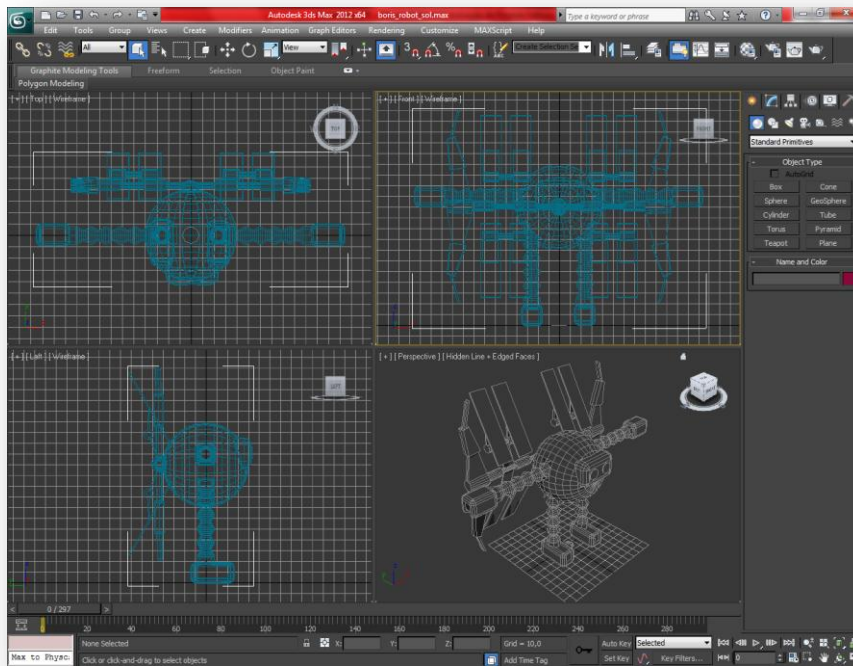


Fig 3. 14 - Model of Boris in solar mode in Autodesk 3Ds Max 2012.

The body of the character already has textures, since the original model of Boris was used with no modifications. However, the solar panels on its back needs to be unwrap (see figure 3.15).

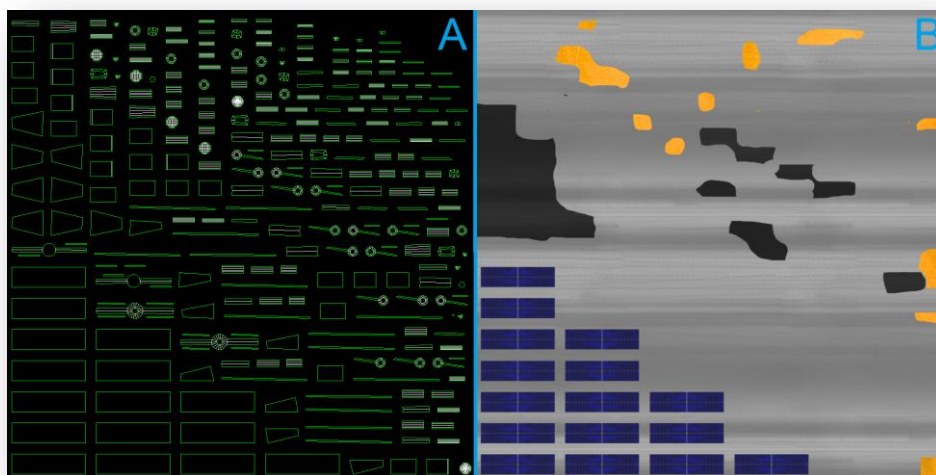


Fig 3. 15 - A: UVW map of the model of Boris in solar mode; B: UVW map of the model of Boris in solar mode with textures.

Once the textures were finish, they were applied to the model, giving us the final version of the character (see figure 3.16).

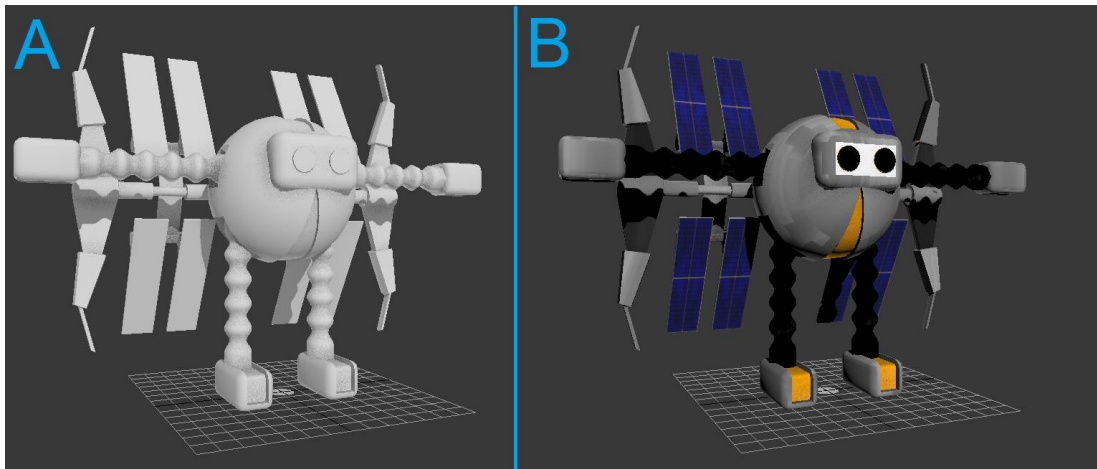


Fig 3. 16 - A: Model of Boris without textures; B: Model of Boris with textures.

Since we had new parts to Boris, it was necessary to add new bones to the created biped in order to have envelopes over the vertex on the solar panels (see figure 3.17). Once the bones and envelopes were in place, it was possible to create the animations for solar mode.

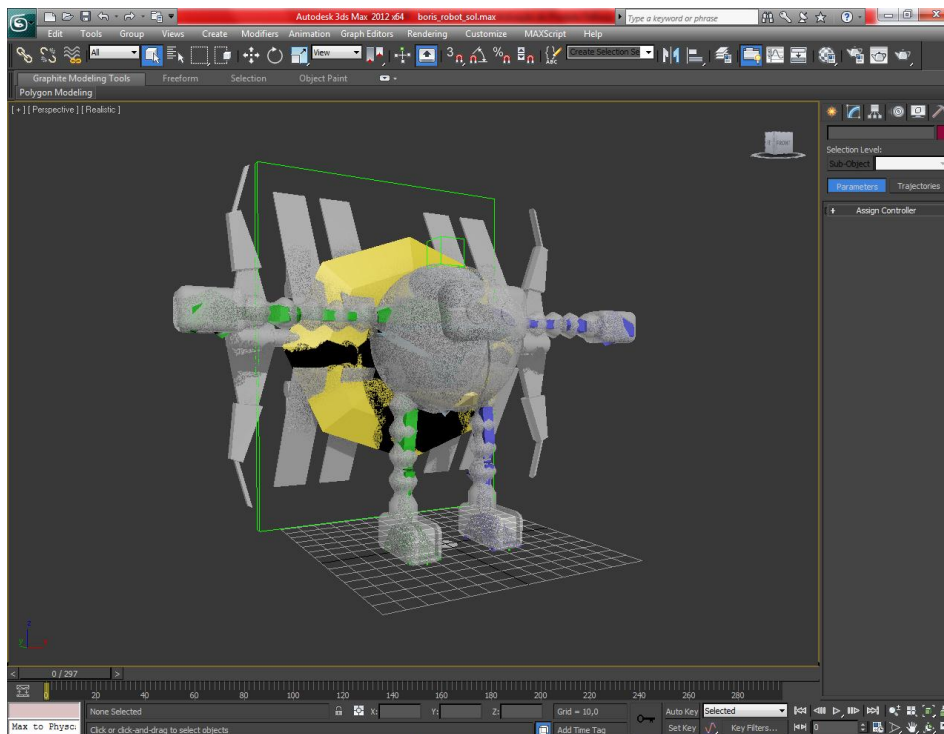


Fig 3. 17 - Model of Boris in solar mode with biped inserted.

During solar mode, Boris mobility will be reduced, this means that the 3D model will only have the following animations:

- Run;
- Run backwards;
- Idle;
- Recharge;

In order to give a more realistic look to the character in all its forms, various normal maps were created based on its texture files (see figure 3.18). To generate the normal images of the textures, it was used NVIDIA Texture Tools for Adobe Photoshop [100].

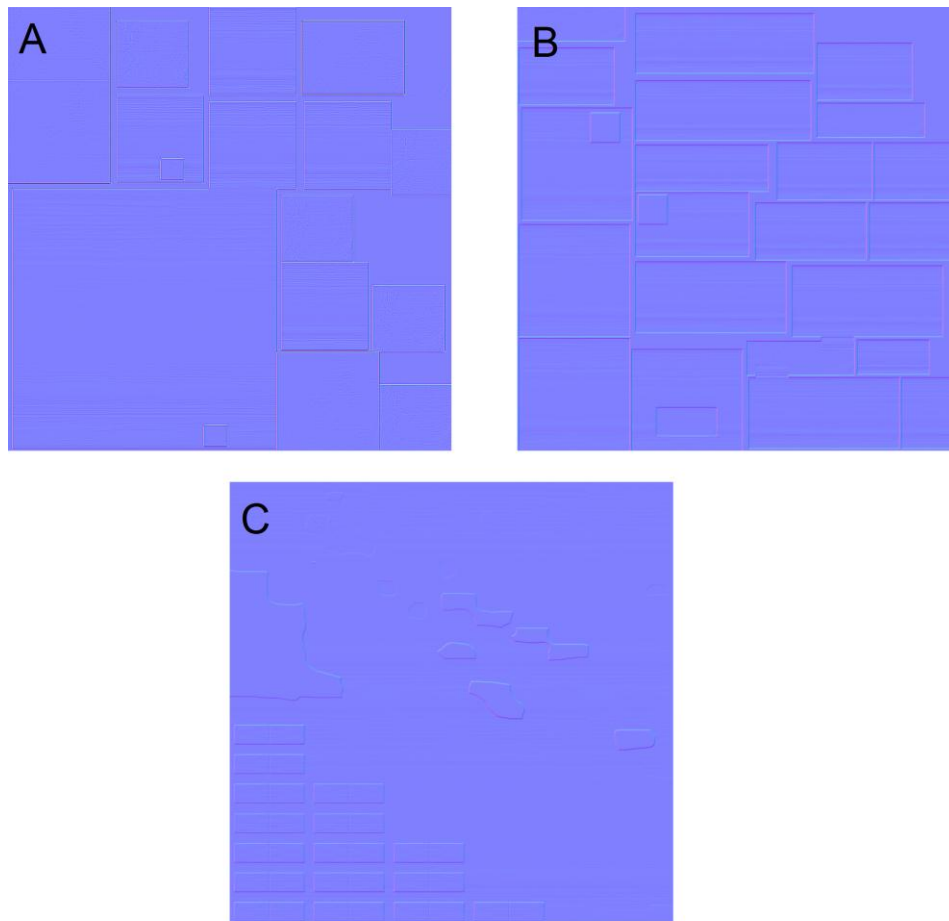


Fig 3. 18 - A: Normal map of Boris in robot mode; B: Normal map of Boris in sphere mode; C: Normal map of Boris in solar mode.

3.3.2. Kate

Now that the main character of the game is ready, is now time to take care of the other characters, and the first one we are going to see is Kate (see figure 3.19).



Fig 3. 19 - Concept of Kate given by the concept artist.

Kate is a 16 year-old girl that lives in Cypricene Island. She is very sick and is too weak to go to the medical center. Since she is alone, she decides to build Boris, a small droid that she sends in search of help.

This is one of the characters that the player will find during the game and which who he will have to interact in order to receive new quests, upgrades or information's.

Just like Boris, the model of Kate was modeled on 3Ds Max using a model sheet given by the concept artist (see figure 3.20).

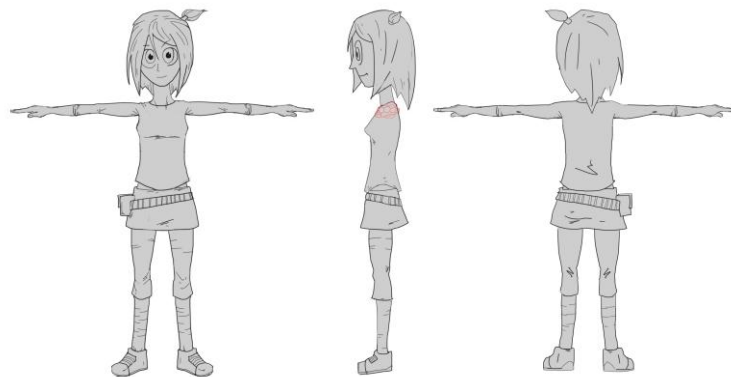


Fig 3. 20 - Model sheet of the character Kate.

Using the model sheet was created the 3D model of Kate (see figure 3.21 A). To create the texture to the model was used the UVW unwrap. The created textures were then applied to the 3D mesh (see figure 3.21 B).

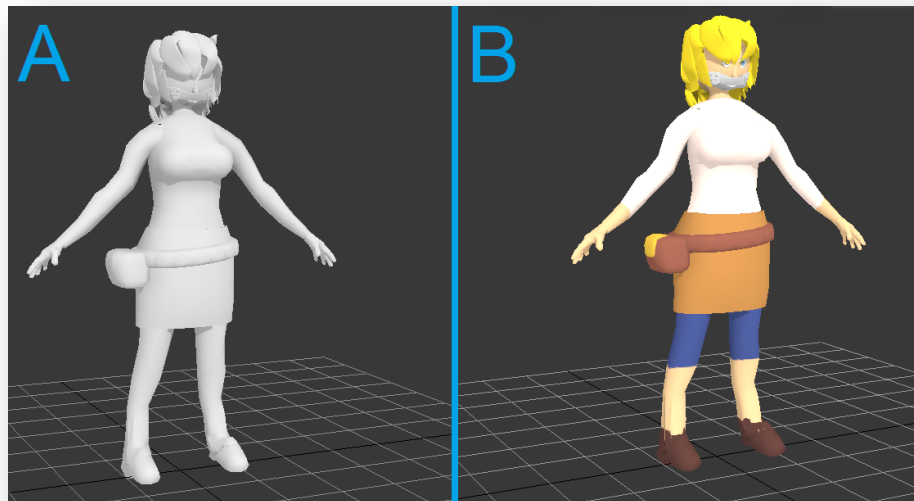


Fig 3. 21 - A: Model of Kate without textures; B: Model of Kate with textures.

With Kate mesh finished, is now time to insert the biped on the mesh and use the skin modifier to connect both (see figure 3.22 A). Once that task is over, it's time to adjust the envelopes of Kate's bones before we can start animating the model (see figure 3.22 B).

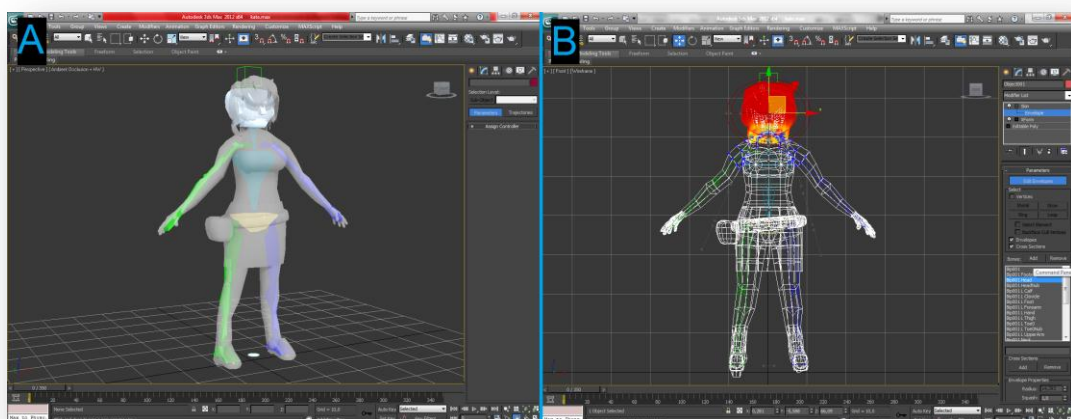


Fig 3. 22 - A: Model of Kate with biped inserted; B: Adjusting the envelopes of Kate model.

Since Kate is a non-playable character, it won't need so many animations as Boris model had. Basically, this model only needs two animations, since it will only have two behaviors inside the game:

- Idle;
- Talk;

Once again we use key-frame animations to animate the mesh. After that, the only work remaining is to export the model to FBX and the generated FBX to X. The mesh and the textures can now be importer into XNA.

3.3.3. Dr. Jacob

Other character in the game is Dr. Jacob (see figure 3.23).



Fig 3. 23 - Concept of Dr. Jacob given by the concept artist.

Dr. Jacob is the one in charge of the medical center of Cypricene Island. He has spent the last decade studying the Stigma, a new disease that has been affecting mankind. Dr. Jacob struggles every day to find a cure to this new plague.

This is one of the characters that the player will find during the game and which who he will have to interact in order to receive new quests, upgrades or information's.

Just like the previous models, the model of Dr. Jacob was modeled on 3Ds Max using a model sheet given by the concept artist (see figure 3.24).



Fig 3. 24 - Model sheet of the character Dr. Jacob.

Using the model sheet was created the 3D model of Dr. Jacob (see figure 3.25 A). To create the texture to the model was used the UVW unwrap. The created textures were then applied to the 3D mesh (see figure 3.25 B).

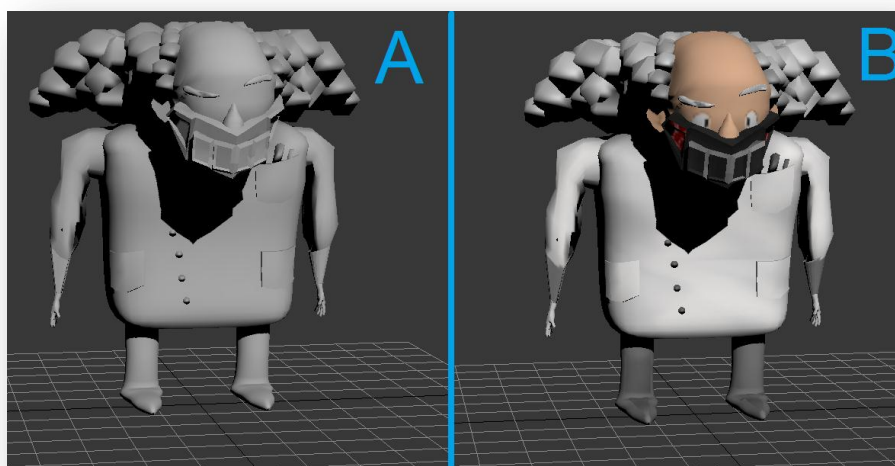


Fig 3. 25 - A: Model of Dr. Jacob without textures; B: Model of Dr. Jacob with textures.

With the mesh created, is now time to insert the biped on the mesh and use the skin modifier to connect both (see figure 3.26 A). Once that task is over, it's time to adjust the envelopes of Dr. Jacob's bones before we can start animating the model (see figure 3.26 B).

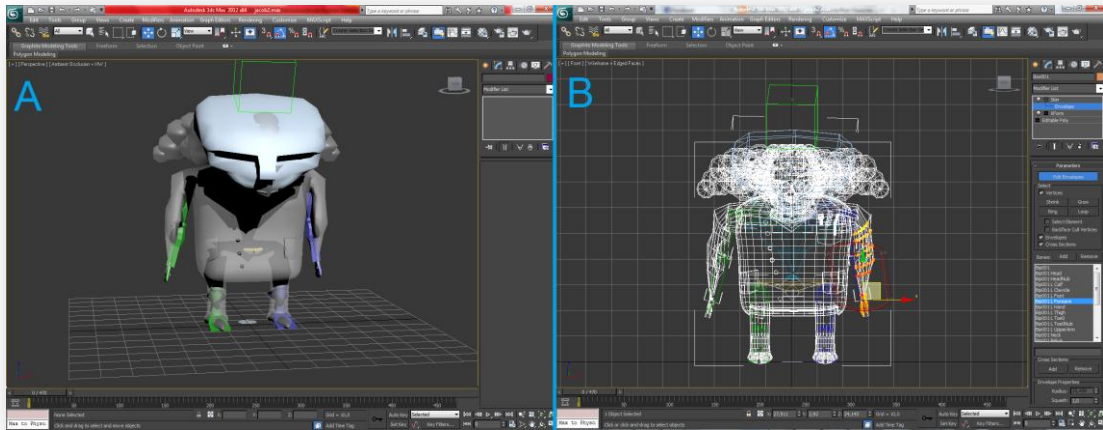


Fig 3. 26 - A: Model of Dr. Jacob with biped inserted; B: Adjusting the envelopes of Dr. Jacob model.

Since Dr. Jacob is a non-playable character, it just needs the same type of animations as Kate did. Basically, this model only needs two animations, since it will only have two behaviors inside the game:

- Idle;
- Talk;

3.3.4. Tom A. Toe

Tom A. Toe is another character in the game (see figure 3.27).



Fig 3. 27 - Concept of Tom A. Toe given by the concept artist.

Tom A. Toe is the engineer responsible for the wastewater treatment plant of the island. He may be a little grumpy, but he is an expert on renewable and clean energies.

The player will find Tom on the third level, and he is will have to interact with him order to receive new quests, upgrades or information's.

The development process of this character was identical to the other characters. 3Ds Max was used for the modeling using a sheet given by the concept artist (see figure 3.28).



Fig 3. 28 - Model sheet of the character Tom A. Toe.

Using the model sheet was created the 3D model of Tom (see figure 3.29 A). Like on the previous characters, was used UVW unwrap for texturing (see figure 3.29 B).

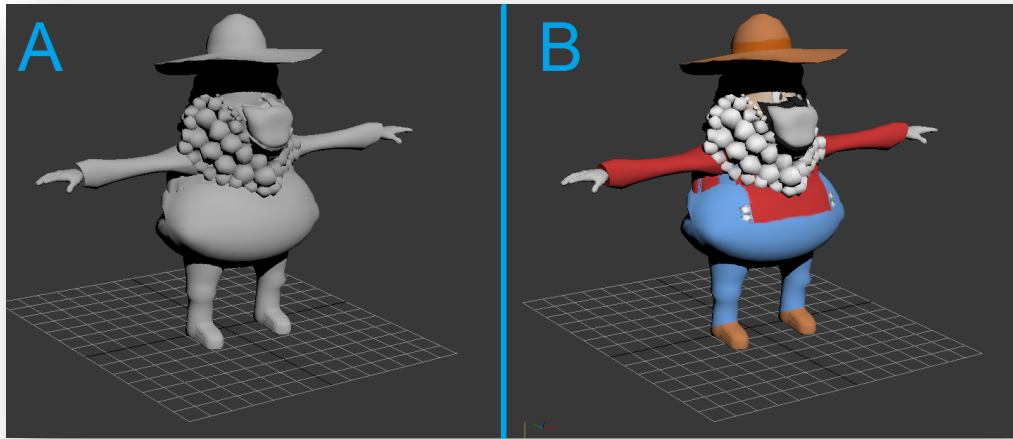


Fig 3. 29 - A: Model of Tom A. Toe without textures; B: Model of Tom A. Toe with textures.

The skinning of the character followed the same steps as the other game characters (see figure 3.30).

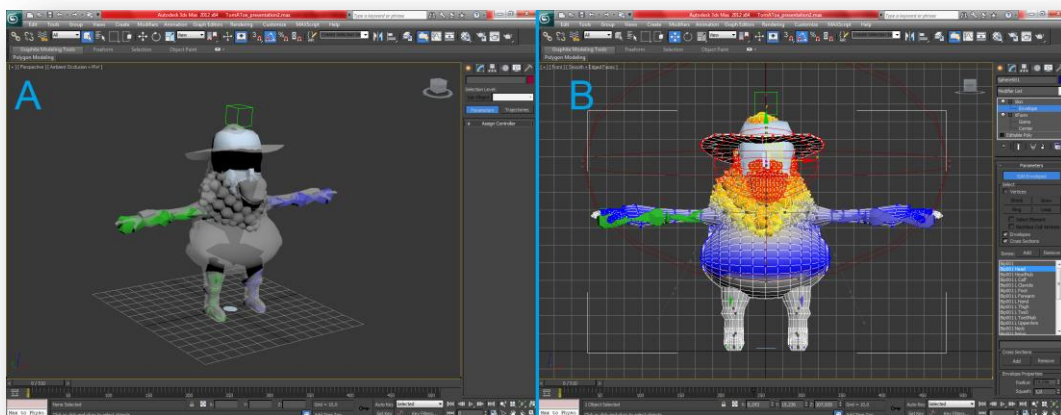


Fig 3. 30 - A: Model of Tom with biped inserted; B: Adjusting the envelopes of Tom model.

Tom has the same type of animations as Kate and Jacob did:

- Idle;
- Talk;

3.4. Terrain Models and Level Assets

In this section we will present the terrains developed for the three levels of the game, and the level assets created for each level, like buildings, trees and collectable items. We will start by presenting the terrains used on the game levels, which are the base of every level in the game. All game levels and assets were created using Autodesk 3Ds Max 2012 [97] and Adobe Photoshop CS5 [98].

3.4.1. Terrain Models

Clean World is composed by three levels: two levels on the outside world, and one level inside a cave. Three terrains were created in order to make the game more attractive to the player.

Level one and three are located in the outside world and represent an island. In order to create a more realistic illusion to the player, giving him the idea of being on an island, both terrains were modeled together in a single model (see figure 3.31).

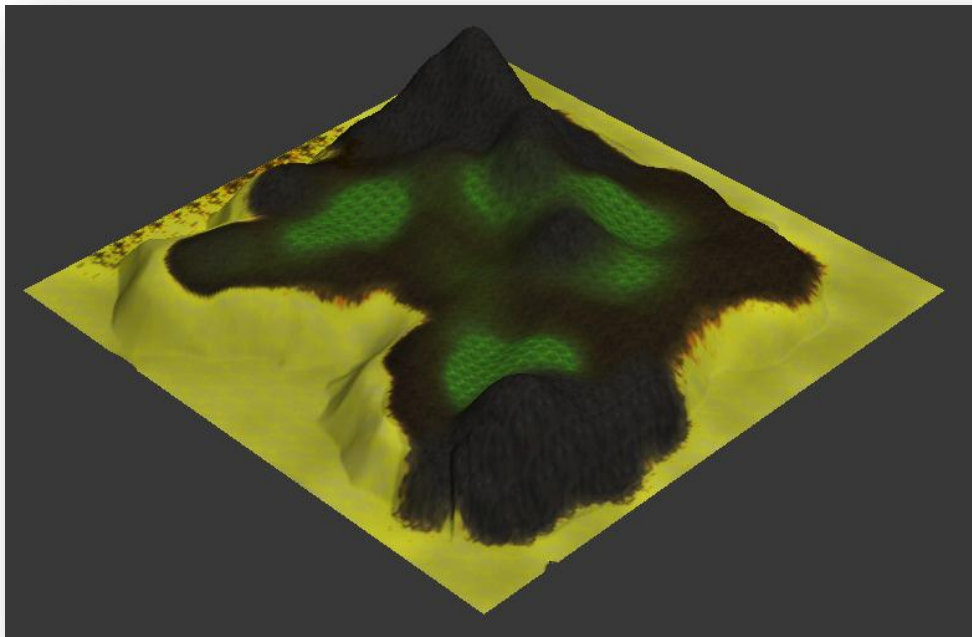


Fig 3. 31 - Complete model of the Cypricene Island.

However, to make the game run faster, the whole island was separated in two different models. Since the player can't access the third level without going through the cave (the second level), doesn't make any sense to load all the geometry on one level if it's not used. The model of the island was then divided into two different models (see figure 3.32), one for each level.

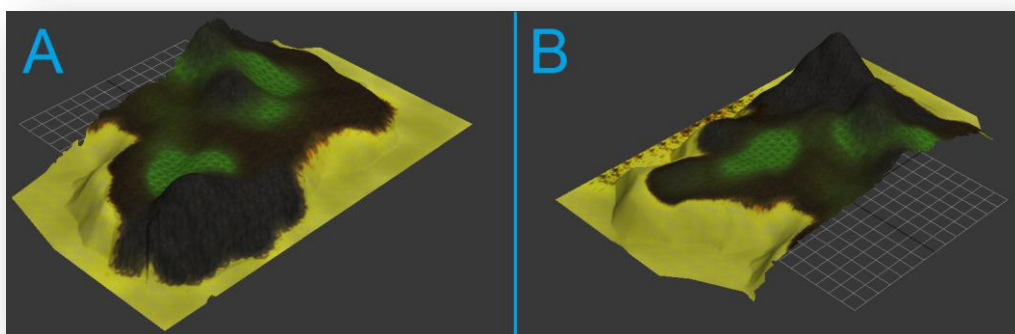


Fig 3. 32 - A: Terrain of level one; B: Terrain of level three.

The second level of the game is played inside a cave, which means that a new terrain had to be created for this level. The cave wall were modeled through the use of several extrudes from a tube primitive on 3Ds Max. Once the wall were created, it was modeled the floor of the cave, which was later combined with the walls. Some machinery was also modeled inside the cave. Once everything was modeled, all 3D meshes were texturized with uvw unwrap, giving us the final look of the level (see figure 3.33).



Fig 3. 33 - A: Terrain of level two.

3.4.2. Decorative 3D Assets

Levels are not just terrains. In order to create the desired environment for the level and give a unique identity to it, it's necessary to populate the scenario with all kind of decorative assets. We call decorative assets to all 3D assets that are in the game level but are static (i.e., do not interact with the player except for collision detection). Following are presented several images of the static assets created.

Kate's House

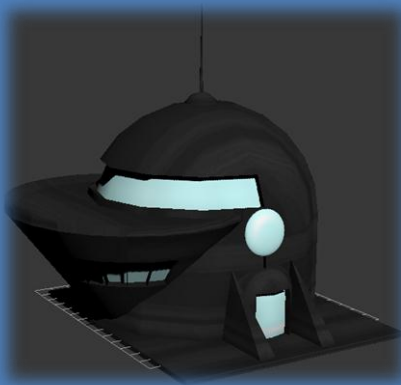


Fig 3. 34 - kate's house model.

Description: This is the model used as Kate's house.

Number of vertex: 2150

Number of faces: 2121

Level: Level 1

Lamp Ring

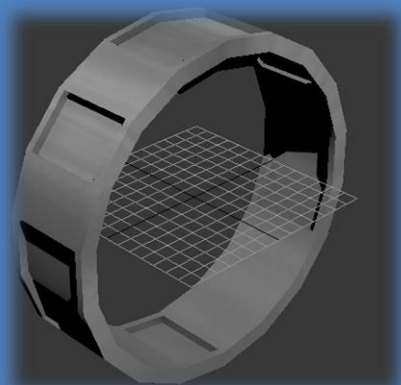


Fig 3. 35 - Lamp ring model.

Description: Lamp ring used on bonus level 2.

Number of vertex: 200

Number of faces: 200

Level: Bonus level 2

Rails

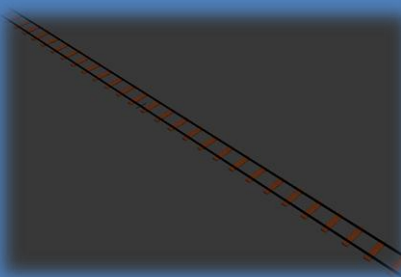


Fig 3. 36 - Rails model.

Description: Rails used on bonus level 2.

Number of vertex: 8880

Number of faces: 16192

Level: Bonus level 2

Medical Center

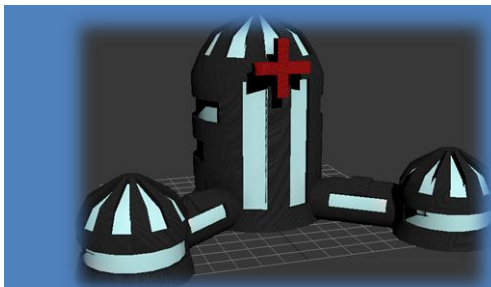


Fig 3. 37 - Medical center model.

Description: Medical center of the Cypricene Island.

Number of vertex: 956

Number of faces: 1908

Level: Level 1

Wood Recycling Machine



Fig 3. 38 - Wood recycling machine model.

Description: Machine used to recycle wood.

Number of vertex: 766

Number of faces: 738

Level: Level 2

Toxic Waste Cleaner



Fig 3. 39 - Toxic waste cleaner model.

Description: Machine used to clean toxic waste.

Number of vertex: 766

Number of faces: 738

Level: Level 1

Wind Tower

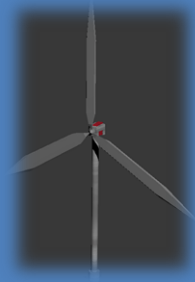


Fig 3. 40 - Wind tower model.

Description: Wind tower user to produce energy.

Number of vertex: 576

Number of faces: 624

Level: Level 3

Lighthouse



Fig 3. 41 - Lighthouse model.

Description: The island lighthouse.

Number of vertex: 878

Number of faces: 834

Level: Level 3

Wall

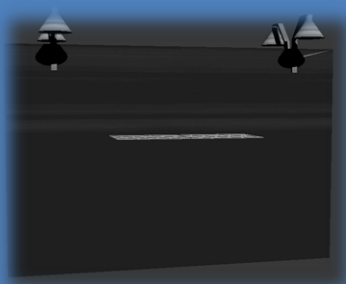


Fig 3. 42 - Wall model.

Description: Wall barrier.

Number of vertex: 424

Number of faces: 385

Level: Level 1 and 3

Platforms for Watermill Mini-game

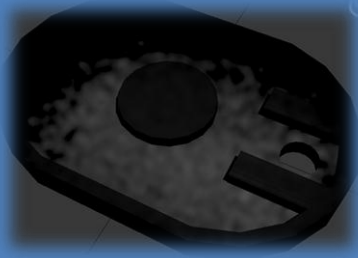


Fig 3. 43 - Platform model.

Description: 6 Platforms for the watermill mini-game

Number of vertex: 329 - 387

Number of faces: 658 - 778

Level: Level 3

Bridge Part

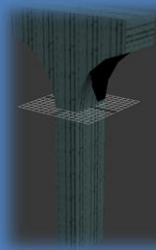


Fig 3. 44 - Part of the bridge model.

Description: Part of the bridge of bonus level 2.

Number of vertex: 88

Number of faces: 75

Level: Bonus level 2

Harbor



Fig 3. 45 - Island harbor model.

Description: The island harbor.

Number of vertex: 2038

Number of faces: 2000

Level: Level 3

Energy Pole

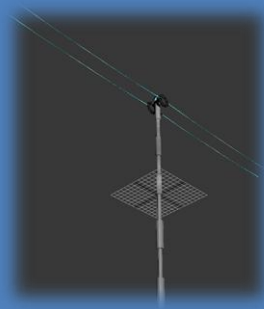


Fig 3. 46 - Energy pole model.

Description: An energy pole.

Number of vertex: 2828

Number of faces: 3616

Level: Level 3

Watermill tower



Fig 3. 47 - Watermill tower model.

Description: The tower of the watermill.

Number of vertex: 2996

Number of faces: 5526

Level: Level 3

Tunnel Section

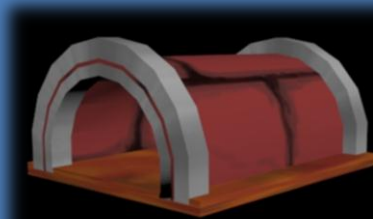


Fig 3. 48 - Tunnel section model.

Description: A tunnel section.

Number of vertex: 405 - 438

Number of faces: 671 - 772

Level: Bonus level 1

Mining Walker



Fig 3. 49 - Mining walker model.

Description: A mining walker.

Number of vertex: 1314

Number of faces: 1374

Level: Level 2

Watermill

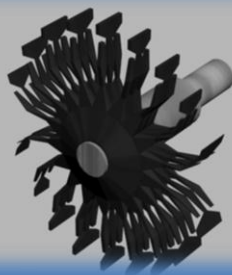


Fig 3. 50 - Watermill model.

Description: The watermill.

Number of vertex: 1008

Number of faces: 976

Level: Level 3

Wastewater Treatment Plant

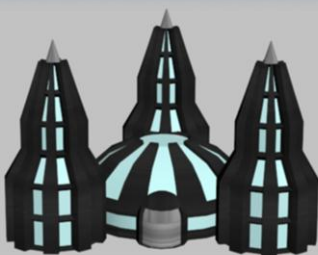


Fig 3. 51 - WWTP model.

Description: The wastewater treatment plant.

Number of vertex: 1795

Number of faces: 1730

Level: Level 3

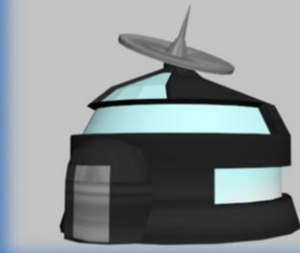
Tom A. Toe House

Fig 3. 52 - Tom's house model.

Description: Tom's house.**Number of vertex:** 868**Number of faces:** 854**Level:** Level 3**3.4.3. Interactive and Collectable 3D Assets**

Spread across the levels, there are several 3D objects that the player can interact with and/or collect. Following we will present these objects, which can be items or even control consoles for manipulating the game scenario.

Question Mark

Fig 3. 53 - Question mark model.

Description: Every time the player catches a question mark on the tutorial, a tip is given.**Number of vertex:** 736**Number of faces:** 1464**Level:** Level 1

Recycling Machine



Fig 3. 54 - Recycling machine model.

Description: Machine used to recycle garbage.

Number of vertex: 1586

Number of faces: 3144

Level: Level 1 and 2

Recycling Container



Fig 3. 55 - Yellow recycling container model.

Description: There are several of these containers, one for each type of garbage, each one with different colors. In total, there are 8 different colors of containers.

Number of vertex: 272

Number of faces: 270

Level: Level 1 and 2

Arrow



Fig 3. 56 - Arrow model.

Description: Indicates the path to the player during the game

Number of vertex: 20

Number of faces: 18

Level: Level 1, 2 and 3

Recycling Platform

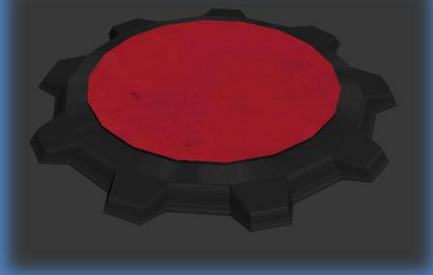


Fig 3. 57 - Recycling platform model.

Description: There are several of these platforms, one for each type of garbage, each one with different colors. In total, there are 8 different colors of platforms.

Number of vertex: 108

Number of faces: 92

Level: Level 1 and 2

Solar Platform

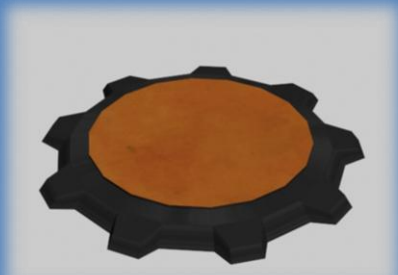


Fig 3. 58 - Solar platform model.

Description: These platforms are used to activate machines using the solar panels of Boris.

Number of vertex: 108

Number of faces: 92

Level: Level 1 and 3

Recharger



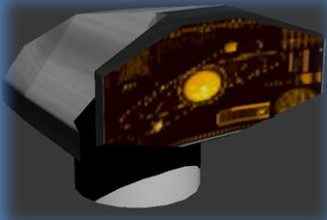
Fig 3. 59 - Recharger model.

Description: These machines are used to recharge Boris batteries inside the cave.

Number of vertex: 404

Number of faces: 392

Level: Level 2

Console Terminal	
 Fig 3. 60 - Console terminal model.	Description: These terminals are used to activate door or machines.
	Number of vertex: 82
	Number of faces: 114
	Level: Level 1, 2 and 3

There are also several collectable items (for example, figure 3.61 shows a syringe model) that were modeled and that the player can catch in all the levels. The following list presents the items created for the game:

- Paper box 1;
- Paper box 2;
- Glass bottle;
- Glass cup;
- Plastic bottle;
- Milk package;
- Can;
- Battery;
- Newspaper;
- Plastic taparuere;
- Syringe;
- Rotten apple;
- Package with toxic waste;
- Wood pallet
- Wind tower engine;
- Wind tower pole;
- Wind tower helix;
- Wind tower base;

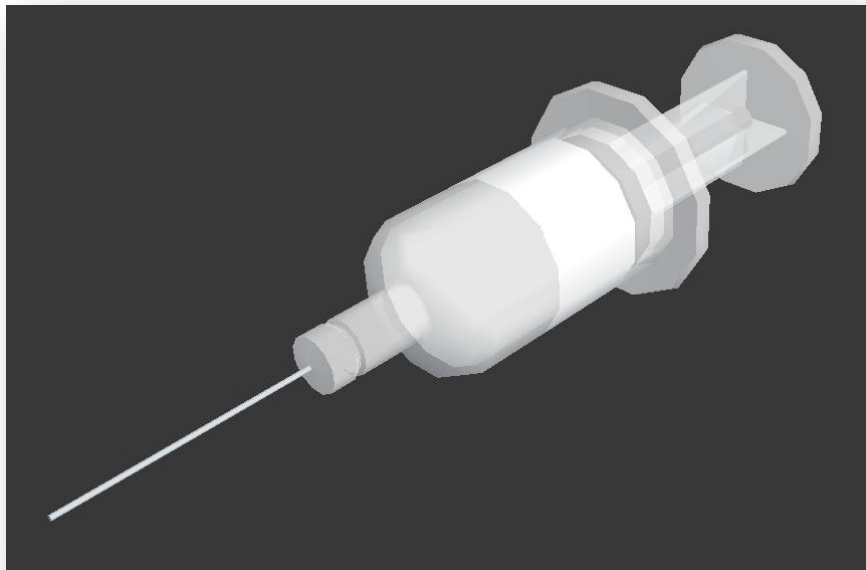


Fig 3. 61 - The syringe collectable item model.

3.5. Level Design

To create the levels of the game, it was used the SunBurn [101] level editor, this was the only development tool used to create the game. In this section will be described the process of creating the final visual look of the levels and models in the game.

Using SunBurn, it was created a 3D scene in other to create the level. Once the engine creates the 3D scene for the level, it's time to import all assets of that level to SunBurn. This is done by indicating the location of the files in the directory of the XNA game project. The engine will then load the 3D meshes and the textures in order to prepare then to the scene.

With all the content loaded, it's now time to start building the levels of the game. The first asset to be added is the terrain, however, since we are using DigitalRune engine for physics, and the terrain model is a triangle mesh, this means that the collision with terrain and the other 3D assets will be done by calculating collisions with the faces of the terrain mesh. Thus

the terrain must be added by code. This happens because DigitalRune and SunBurn are two separated engines with no connection.

With the terrain placed in the 3D scene, we can now add the static meshes that belong to the level. This process is done visually using the SunBurn level editor, by dragging the desired 3D model to the correct place and dropping it into the scene (see figure 3.62). The static meshes can then be moved, rotated or scaled.

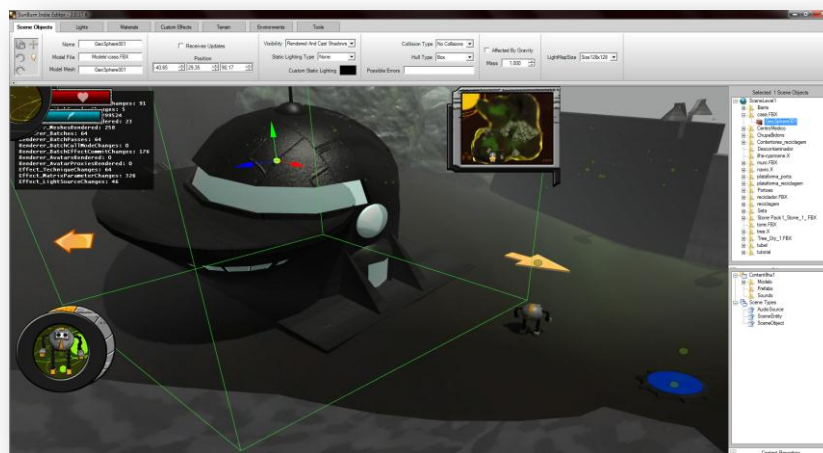


Fig 3. 62 - Positioning a 3D model on SunBurn editor.

All assets of the level must be placed, scaled, and rotated one by one through this process. To give a general idea of the work involved in building the levels, beside the buildings and other visual marks that excel from the landscape, a total of 55 trees and 59 rocks were hand placed only on level one.

With all static models placed, we can advance to illuminating the scene. SunBurn included four different types of light that can be used: ambient, directional, point and spot. Two lights were used to illuminate the scene, one ambient and one directional. The ambient light (see figure 3.63), which is default to every scene created with SunBurn, was configured with the intensity of 0.3 (i.e., a little dark) and with the RGB color: 147, 145, 139. By using this configuration it was possible to create

the idea of a foggy and polluted look that we wanted for the game world in the first level.



Fig 3. 63 - Changing the color of the ambient light.

The directional light simulates the sun (see figure 3.64). This light has a direction, and it was set to cast shadow over all objects on the 3D scene. The intensity of the light was defined with the value of 1.6, and with the RGB values (91, 91, 86) respectively. Other properties of the directional light, such as shadow quality, primary bias and secondary bias were left on their default values. The options single pass rendering, receive updates and light mapped were left disabled.

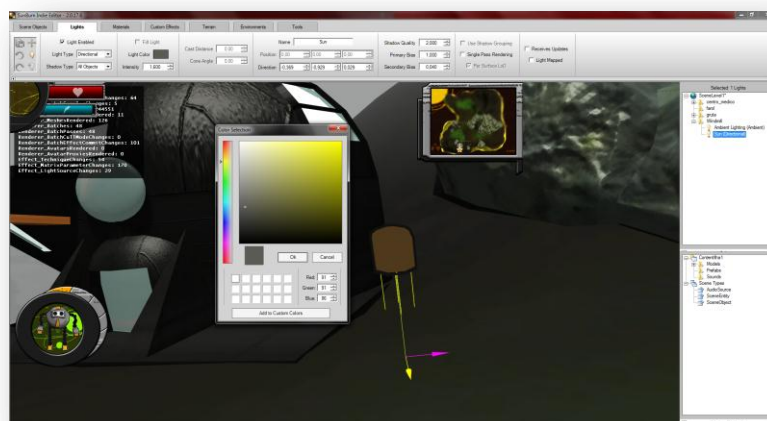


Fig 3. 64 - Changing the color of a directional light.

These two lights are the main lights of the level; however, some other lights were added in certain points to create an environment with realistic look and beauty. On example of such lights is the one placed on the red cross on the top of the medical center (see figure 3.65). This light is a point light RGB (255, 5, 5), with the intensity value of 1, cast distance of 5, and casting shadows over all objects. All other options of the light were left on by default.

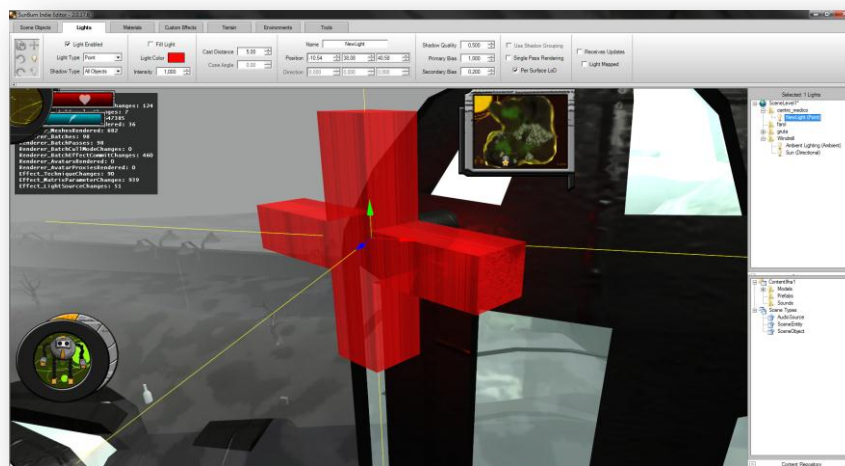


Fig 3. 65 - Changing the settings of a point lighth.

After placing all light and defined their setting, it's time to enhance the look of the level by working on the materials of the 3D objects. When 3D models are loaded to the engine, their material just contains the information about their diffuse map. However, SunBurn allows the users to add normal maps to the materials, as well as change the settings of the diffuse color, emissive color, specular power, and specular amount and transparency. For example, for metal materials, was applied a normal map (see figure 3.66A), and changed the default settings of the specular power and amount to 32 and 1 respectively (see changes can be seen on figure 3.66B and 3.66C).



Fig 3. 66 - A: Normal map used on metallic surfaces; B: Kate's house without normal map; C: Kate's house with normal map.

The final adjustment to the level was the inclusion of a fog with RGB (158, 161, 160) respectively. The properties fog start distance, fog end distance and viewable distance were defined with the values of 20, 55, and 55 respectively. These settings allow the creation of a dense fog over the first level, giving to the player the sensation that the world is covered by a veil of smoke, as it would be expected on a polluted world.

As soon as the scene was finished, the programmers added by code the NPCs and animated meshes and created the game logic for the level.

This process was repeated to all three levels, only changing the values of the materials, lights and fog according to the needs of the desired environment.

3.6. 2D Assets

Beside the 3D assets of the game, it's necessary to create 2D assets for the GUI, like the menus, the mode indicator or the health bar. For the creation of the 2D assets it was used Adobe Photoshop CS5.

Two of the created assets were the background for the main menu and for the game over menu (see figure 3.67).

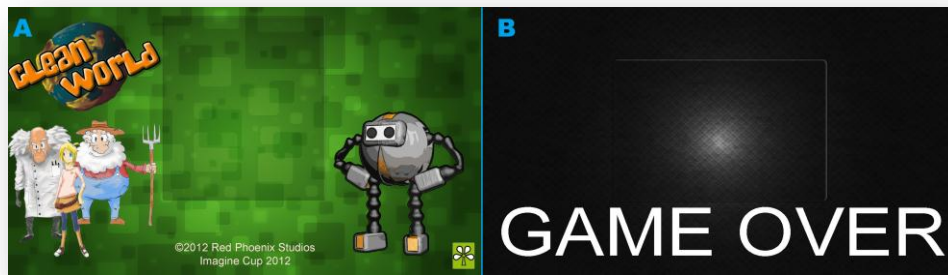


Fig 3. 67 - A: Main menu background; B: Game over menu background.

Eight loading screens were also created. During the game, while the engine loads the content of the level, these loading screens popup, giving some environmental tips and facts to the player. Besides each text with environmental information, one of the NPCs is associated (see figure 3.68).

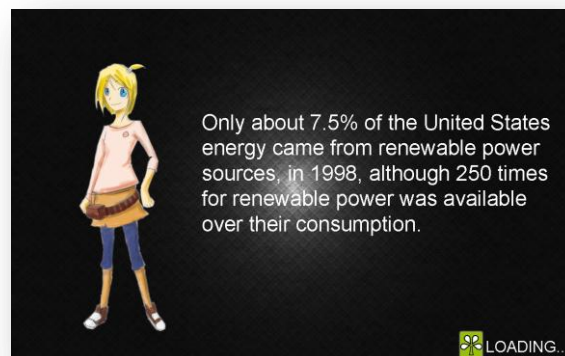


Fig 3. 68 - Example of a loading screen used in the game.

For the GUI of the game had to be developed several sprites, one for the health and energy bars, other for the power mode in use, other for the mini-map, and finally, one for the communicator.

The health and energy bars (see figure 3.69), indicate the levels of health and energy of Boris. The robot spends energy in all his actions, which means that the player needs to recharge his batteries from time to time; otherwise Boris will start to lose health and if the health bar falls to 0, the player will lose the game.

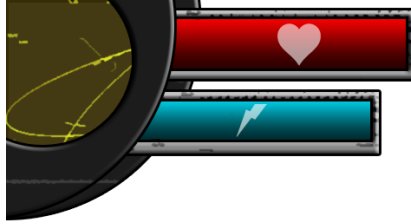


Fig 3. 69 - Health and energy bars sprite.

The power mode indicator (see figure 3.70), lets the player know which mode of Boris is currently active. The player can change through the several power modes during the game.

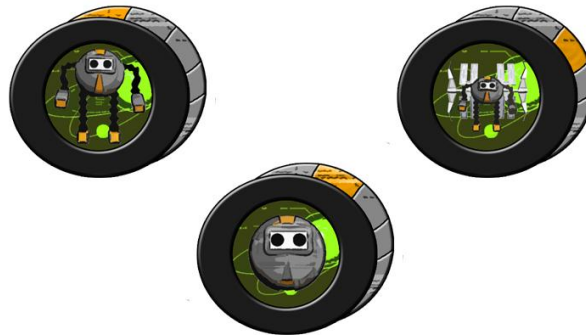


Fig 3. 70 - Boris power mode indicator on its several forms.

The mini-map (see figure 3.71), indicates the player position on the scene, as well as the positions of the active quests. There is a different mini-map for each level.



Fig 3. 71 - Mini map used on level one.

The communicator (see figure 3.72), shows up when the player receives a message from a NPC. There is a different communicator for each

character. For Kate, since the player actions reflect directly on her health, there are 3 sprites of the communicator, each one with a different stage of her health (see figure 3.72).



Fig 3. 72 - Example of the communicator used in the game.

Every time a character interacts with the player, beside the communicator sprite, it's showed a dialog sprite (see figure 3.73). Every single line of dialog in the game had its own sprite, which meant a total of 183 sprites just for dialogs.

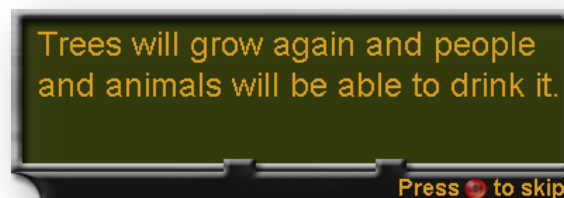


Fig 3. 73 - Example of a dialog sprite used in the game.

Background sprites for the menu system of the game had to be created. The game menu is composed by tabs, meaning that for each tab of the menu, was created a different sprite (see figure 3.74).

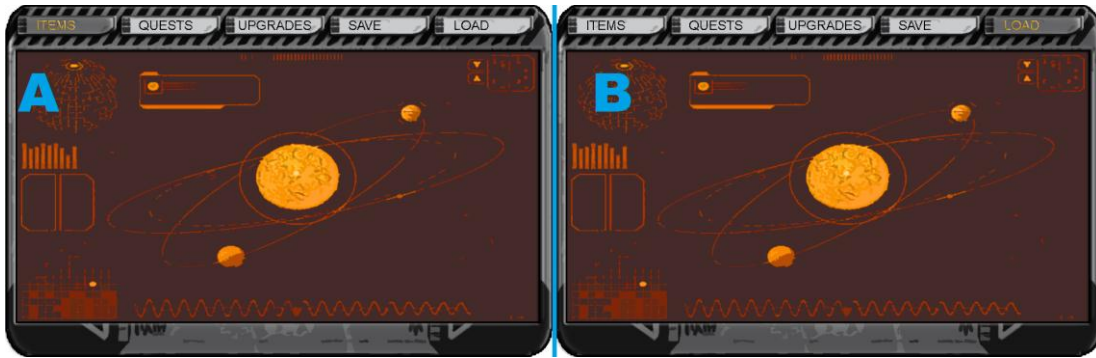


Fig 3. 74 - A: Background of the items tab; B: Background of the load tab.

One of the tabs of the menu is the quest tab. Here, the player can check its active and completed quests. Just like the dialogs, to each description of the quest we have different images (see figure 3.75), which meant a total of 32 sprites.

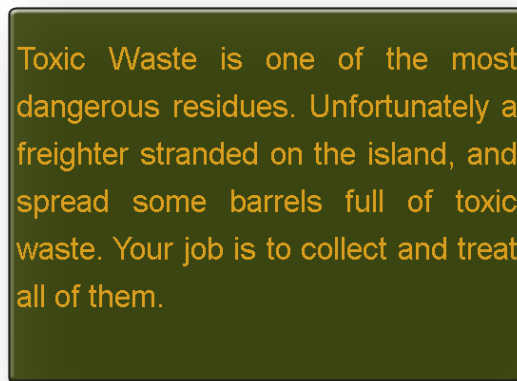


Fig 3. 75 - Example of a quest sprite used in the game.

Beside the sprites for the main game seen before, more than 37 sprites were created for mini-games (see figure 3.76), and 20 sprites for tutorial screens, giving instructions to the player about the main game and mini-games (see figure 3.77).



Fig 3. 76 - Background of one of the developed mini games.



Fig 3. 77 - Tutorial screen of the first bonus level.

Several minor sprites were also produced for feedback messages, or achievement messages spread across the game.

3.7. Summary

In this chapter, Clean World - The 3D Game, we presented the developed work for a serious game with the purpose of helping children to learn important concepts about environmental problems. This game was used as an object of study of the development of games without the use of any major development tools. Section 3.1 described a summarized concept of the game and its development. In Section 3.2 was presented a brief summary of the storyline of the game. Section 3.3 presented the process of developing the game characters, from the concept to the modeling, texturing, and animation. Section 3.4 presented the developed 3D assets used in the creation of the levels of the game. The process of the level design was described on Section 3.5, where the levels were created with the use of SunBurn engine editor. Finally, on Section 3.6 we saw the developed 2D assets for the game, from backgrounds of menus to the creation of sprites to mini-games.

4. Developed Support Applications

4.1. Introduction

Creating a game with no specialized tools support it's a hard and tedious task. During the creation of Clean World, the game described in Chapter 3, several tasks were identified as ideal for the use of specialized tools, in particular the ones related with the storyline or repetitive tasks.

This chapter focuses on the four developed applications created to increase the productivity of the developers of Clean World. The objective of these applications it's to turn repetitive and time consuming task into simple and intuitive processes.

The developed applications are the following:

- Item Manager;
- Quest Manager;
- Dialog Manager;
- Terrain Creator;

In the following sections we are going to present each one of the applications, how they work, how they are connected to the game, and the changes that were done in order to integrate them with Clean World.

4.2. Item Manager

The purpose of this application is to manage all collectable items of the game without being necessary to input item information, for example name or description, on the source code of the game. This tool will let the user create new items, filling all the information needed without a single line of code. The items are saved upon a XML file that can later be loaded to the game.

The Item Manager interface (see figure 4.1), is very simple and intuitive. This tool allows the following operations:

- Create a new item;
- Modify an existing item;
- Delete an existing item;
- Load item file;
- Save item file;

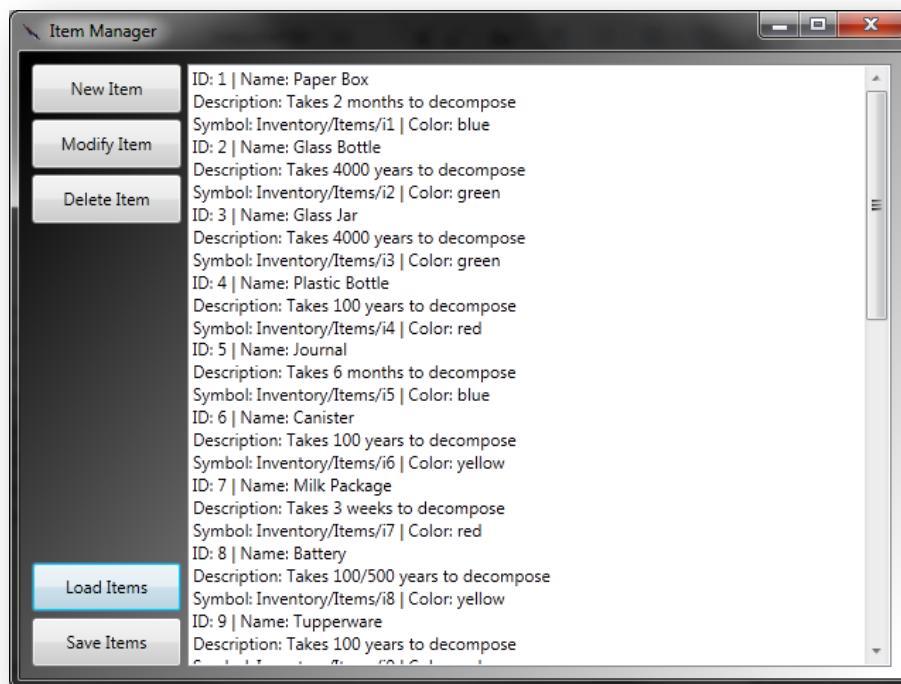


Fig 4. 1 - Screenshot of the Item Manager tool.

On the interface of the tool, there are five buttons, each one relative to one operation. There is also a list, initially in blank, where all created items are showed.

To create a new item, the user simply has to click on the “New Item” button, and a new window will pop-up (see figure 4.2). On this new window, the user has to fill all the data necessary to the creation of a Clean World item.

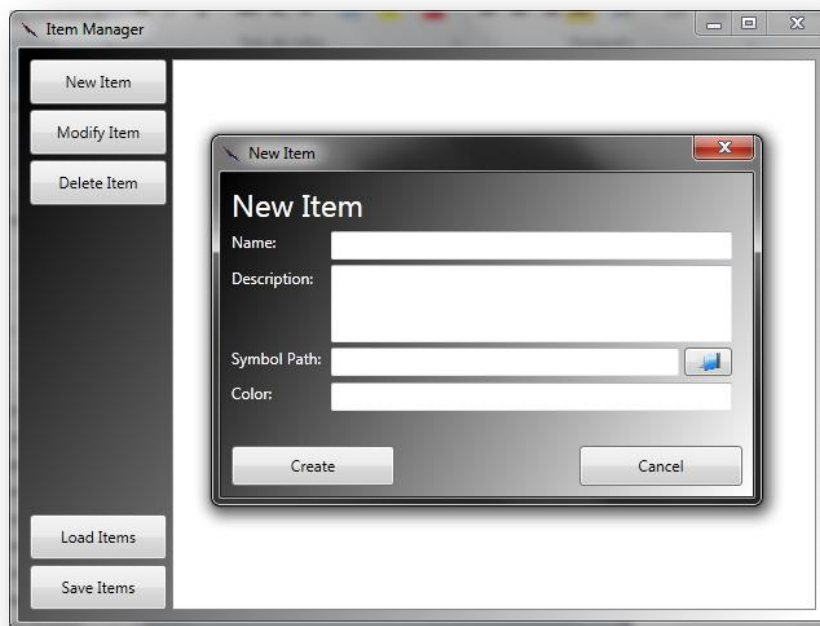


Fig 4. 2 - Creating a new item on Item Manager tool.

All items need the following information to be used in the game:

- Name: the name of the item in the game;
- Description: a brief description of the item that will be showed on the player's inventory;
- Symbol Path: the path indicating the 2D sprite of the item on the game content (note that the 2D sprites had to be previously created and added to the game content);
- Color: this field is used for the recycling mini-games, each color is associated to a specific type of garbage;

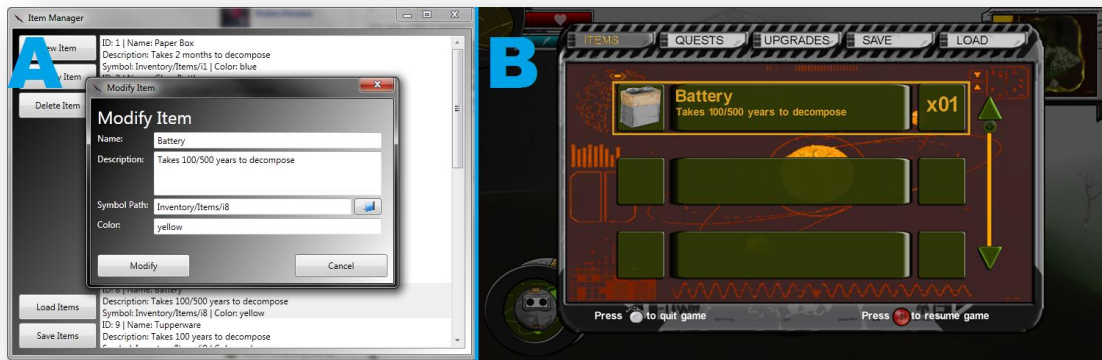


Fig 4. 3 - A: Item information on Item Manager; B: Item information loaded into the game.

If for some reason the user fails to fill all fields, an error message show-up asking the user to fill all information (see figure 4.4).

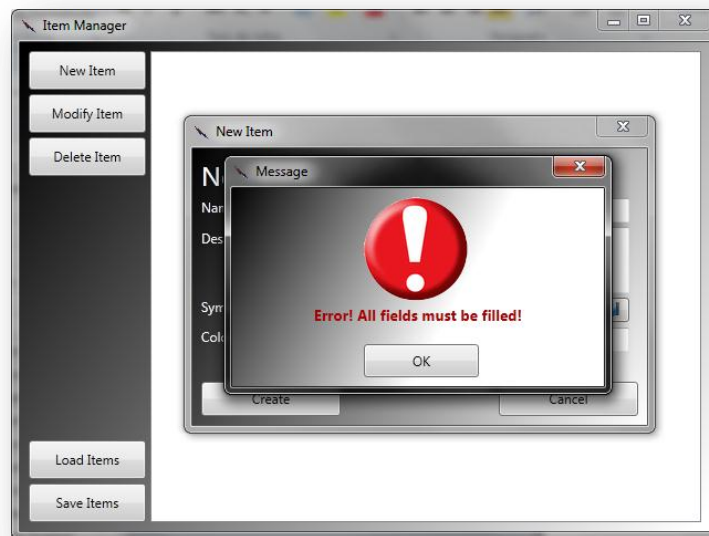


Fig 4. 4 - Screenshot of the error message showing-up.

If all information was inserted, the item is created and the list of items is updated (see figure 4.5).

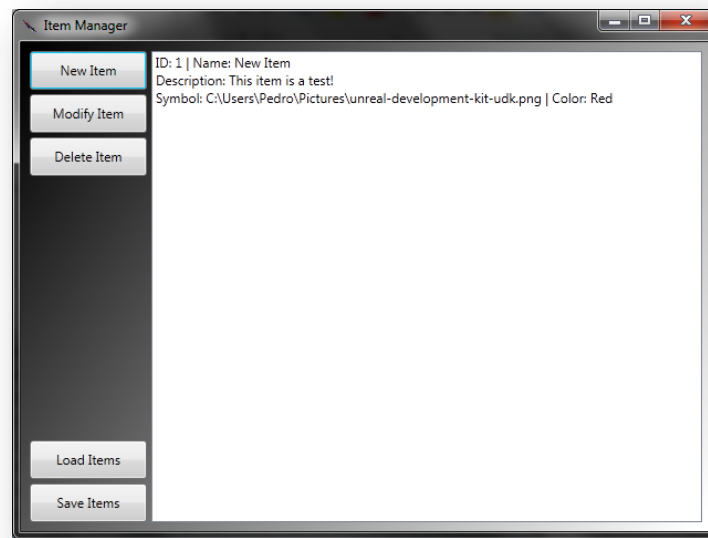


Fig 4. 5 - Screenshot of a new inserted item.

To modify an existing item, the user just needs to select the desired item from the list (see figure 4.6A), and click on the “Modify” button. After that, a window pops-up with all item information (see figure 4.6B).

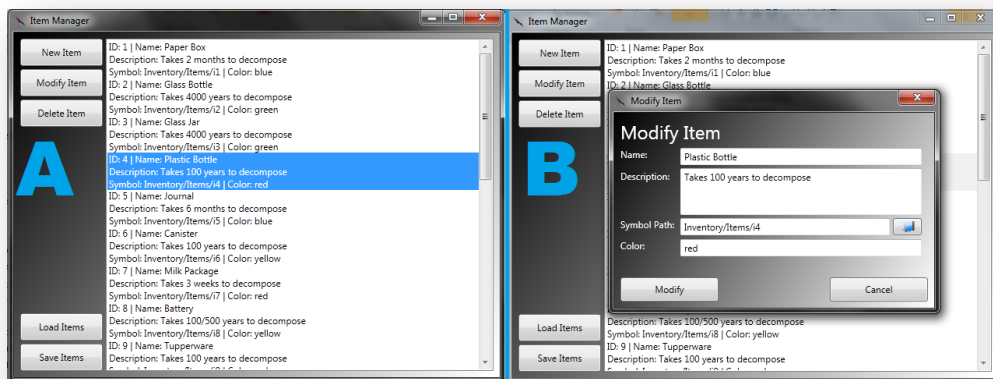


Fig 4. 6 - A: Selecting an item from the list; B: Modifying the item data.

To load an existing file, the user just needs to click on the “Load Items” button, and indicate the location of the file (see figure 4.7).

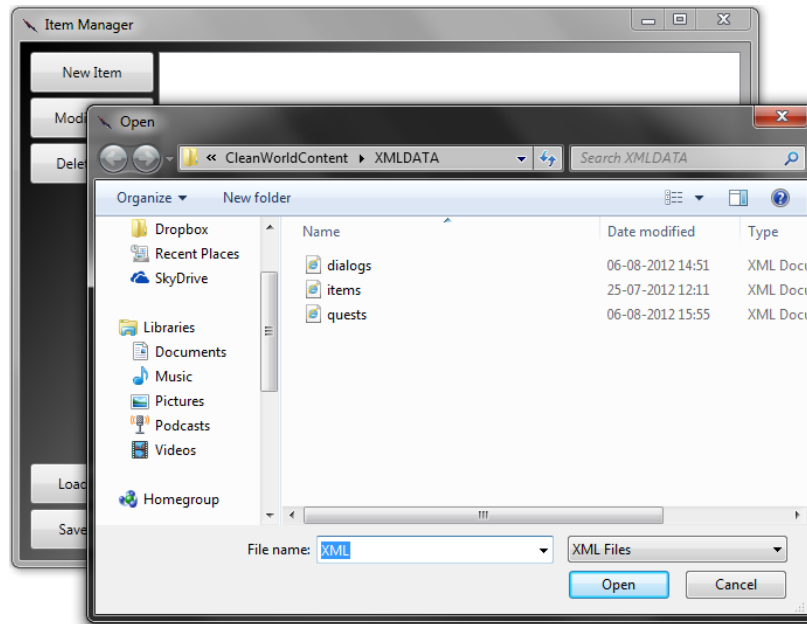


Fig 4. 7 - Opening a saved file with the application.

When the user clicks on the “Save Items” button, all information of the list is saved into an XML file that stores all data. The same XML file is later loaded to the game before the level starts, allowing the engine to load all items on run-time.

The XML file format used for the items is the following:

```
</Items>
  <Item>
    <ID> id </ID>
    <Name> name </Name>
    <Description> description </Description>
    <Symbol> sprite path </Symbol>
    <Color> color </Color>
    <Quantity> quantity </Quantity>
  </Item>
</Items>
```

Note that the quantity field it's not used by the Item Manager

application. Its default value is always 1, and can only be changed inside the game when the player collects more items of the same type. The inclusion of this field on the XML file is only for formal purposes.

Once the file is created, the game developer just needs to save the file into the XMLDATA folder of the game content.

4.3. Quest Manager

The purpose of this application is to manage all game quests without being necessary to create a new sprite to every new quest. This tool will let the user create new quests, filling all the information needed without a single line of code. The quests are saved upon a XML file that can later be loaded to the game.

The Quest Manager interface (see figure 4.8), is very simple and intuitive. This tool allows the following operations:

- Create a new quest;
- Modify an existing quest;
- Delete an existing quest;
- Load quest file;
- Save quest file;

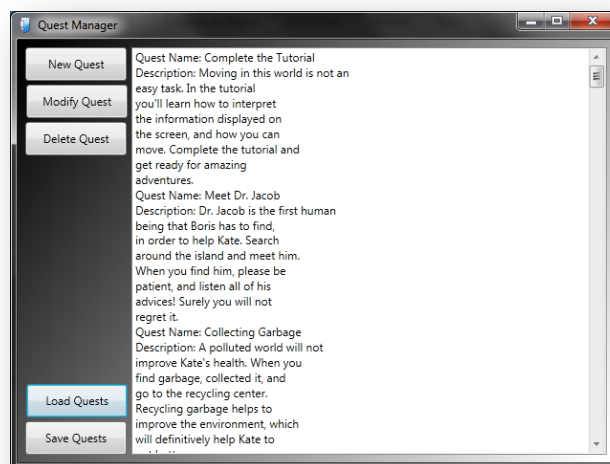


Fig 4. 8 - Screenshot of the Quest Manager tool.

On the interface of the tool, there are five buttons, each one relative to one operation. There is also a list, initially in blank, where all created quests are showed.

To create a new quest, the user simply has to click on the “New Quest” button, and a new window will pop-up (see figure 4.9). On this new window, the user has to fill all the data necessary to the creation of a Clean World quest.

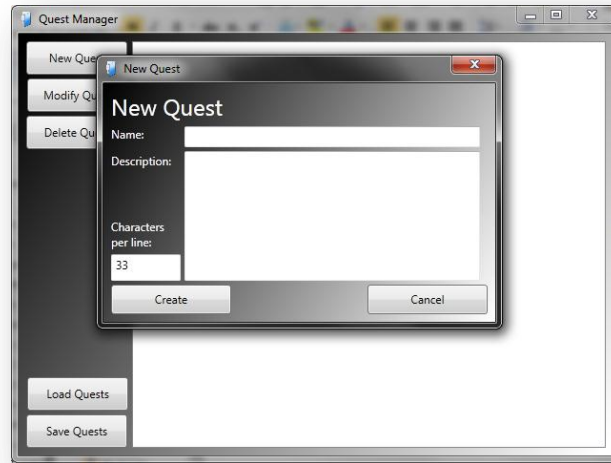


Fig 4. 9 - Creating a new quest on Quest Manager tool.

All quests need the following information to be used in the game:

- Name: the name of the quest in the game;
- Description: a brief description of the quest that will be showed on the player’s quest log;
- Characters per line: it indicates the number of characters per line that should be displayed in the description on the game. The default value is 33;

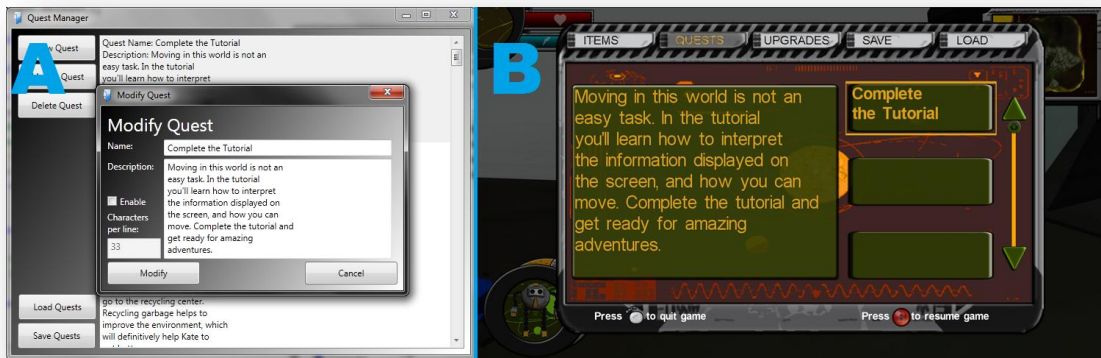


Fig 4. 10 - A: Quest information on Quest Manager; B: Quest information loaded into the game.

If for some reason the user fails to fill all fields, an error message show-up asking the user to fill all information (see figure 4.11).

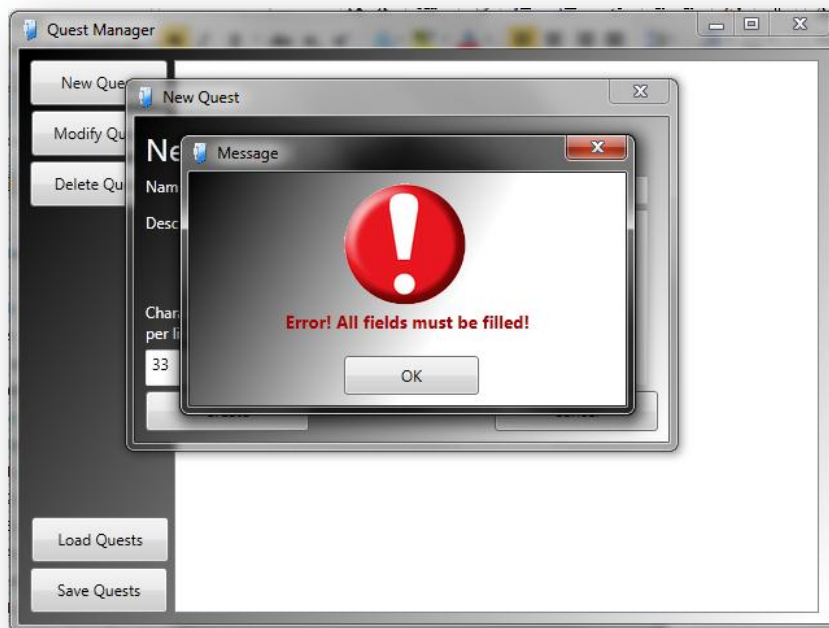


Fig 4. 11 - Screenshot of the error message showing-up.

If all information was inserted, the quest is created and the list of quests is updated (see figure 4.12).

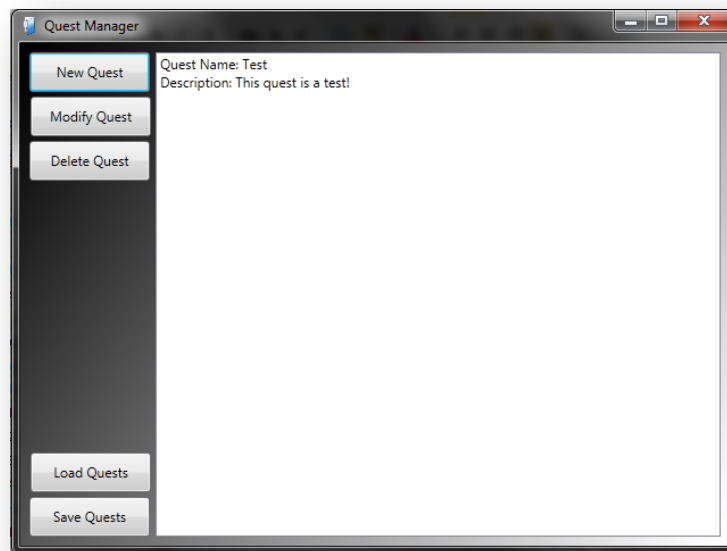


Fig 4. 12 - Screenshot of a new inserted quest.

To modify an existing quest, the user just needs to select the desired quest from the list (see figure 4.13A), and click on the “Modify” button. After that, a window pops-up with all the quest information (see figure 4.13B).

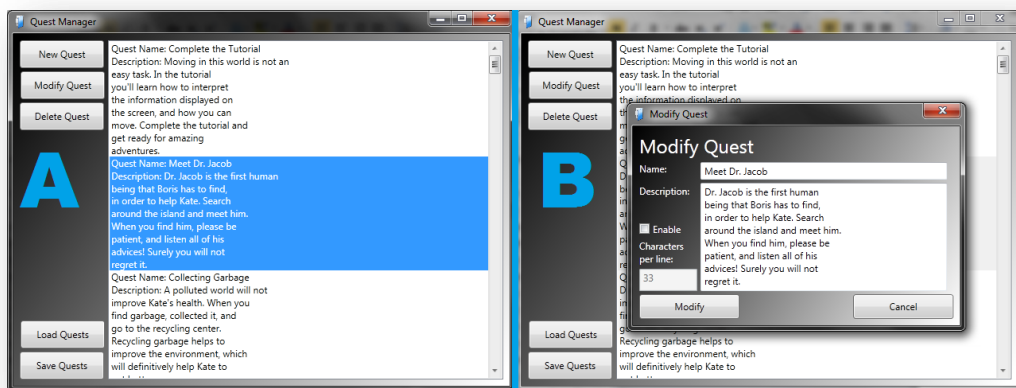


Fig 4. 13 - A: Selecting a quest from the list; B: Modifying the quest data.

To load an existing file, the user just needs to click on the “Load Quests” button, and indicate the location of the file (see figure 4.14).

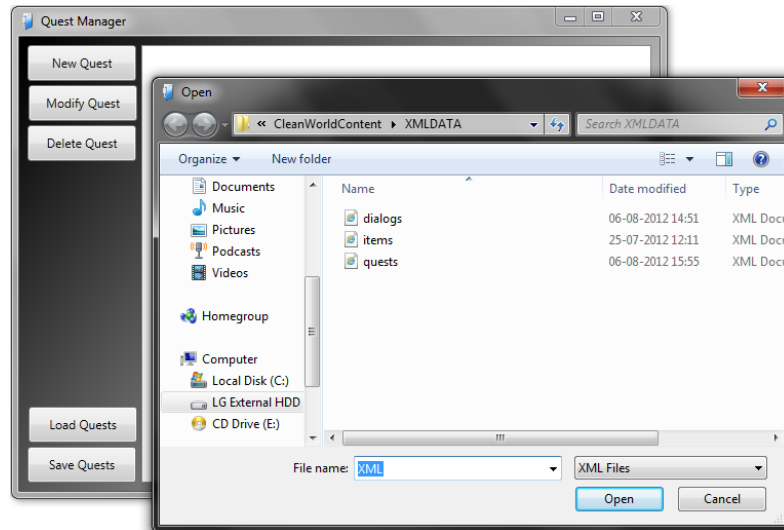


Fig 4. 14 - Opening a saved file with the application.

When the user clicks on the “Save Quests” button, all information of the list is saved into an XML file that stores all data. The same XML file is later loaded to the game before the level starts, allowing the engine to load all quests on run-time.

The XML file format used for the items is the following:

```
<Quests>
  <Quest>
    <Name> name </Name>
    <Description> description </Description>
  </Quest>
</Quests>
```

Note that the character per line information is only used when saving the description information, meaning that the character separation is done every time an quest is created or modified. Therefore, when the description is saved on the XML file, the character separation is already done.

Once the file is created, the game developer just needs to save the file into the XMLDATA folder of the game content.

4.4. Dialog Manager

The purpose of this application is to manage all game dialogs without being necessary to create a new sprite to every new dialog. This tool will let the user create new dialogs, filling all the information needed without a single line of code. The dialogs are saved upon a XML file that can later be loaded to the game.

The Dialog Manager interface (see figure 4.15), is very simple and intuitive. This tool allows the following operations:

- Create a new dialog;
- Modify an existing dialog;
- Delete an existing dialog;
- Load dialog file;
- Save dialog file;

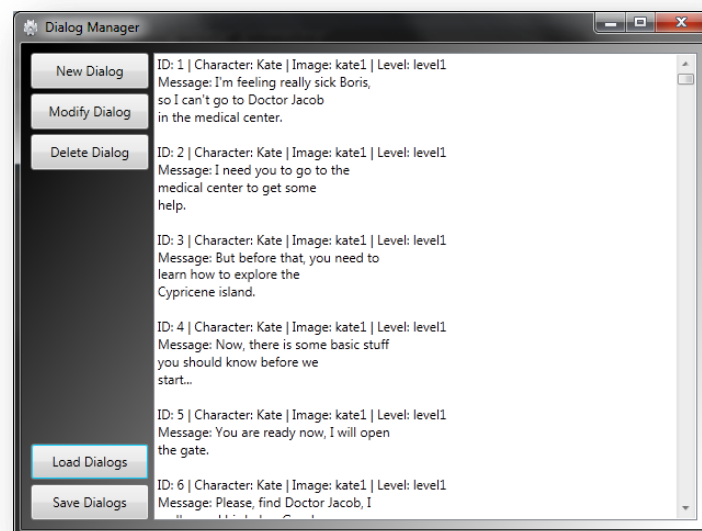


Fig 4. 15 - Screenshot of the Dialog Manager tool.

On the interface of the tool, there are five buttons, each one relative to one operation. There is also a list, initially in blank, where all created dialogs are showed.

To create a new dialog, the user simply has to click on the “New Dialog” button, and a new window will pop-up (see figure 4.16). On this new window, the user has to fill all the data necessary to the creation of a Clean World dialog.

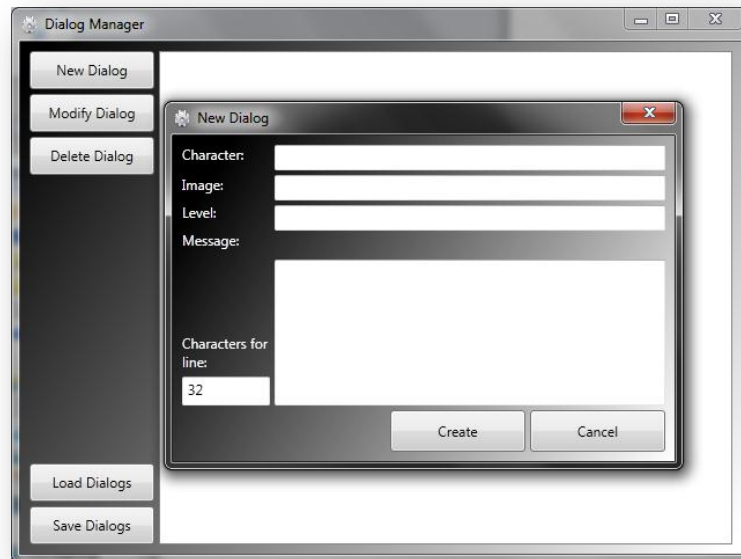


Fig 4. 16 - Creating a new dialog on Dialog Manager tool.

All dialogs need the following information to be used in the game:

- Character: the name of the character that says the dialog in the game;
- Image: the name of the character sprite;
- Level: in which level the dialog is said;
- Message: the text of the dialog;
- Characters per line: it indicates the number of characters per line that should be displayed in the dialog on the game. The default value is 32;



Fig 4. 17 - A: Dialog information on Dialog Manager; B: Dialog information loaded into the game.

If for some reason the user fails to fill all fields, an error message show-up asking the user to fill all information (see figure 4.18).

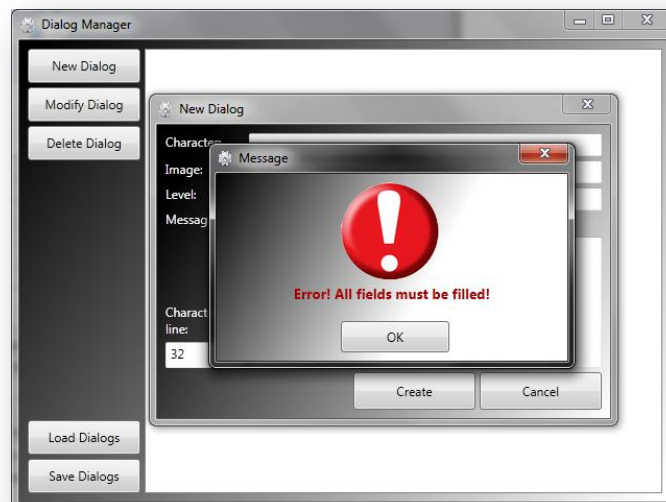


Fig 4. 18 - Screenshot of the error message showing-up.

If all information was inserted, the dialog is created and the list of dialogs is updated (see figure 4.19).

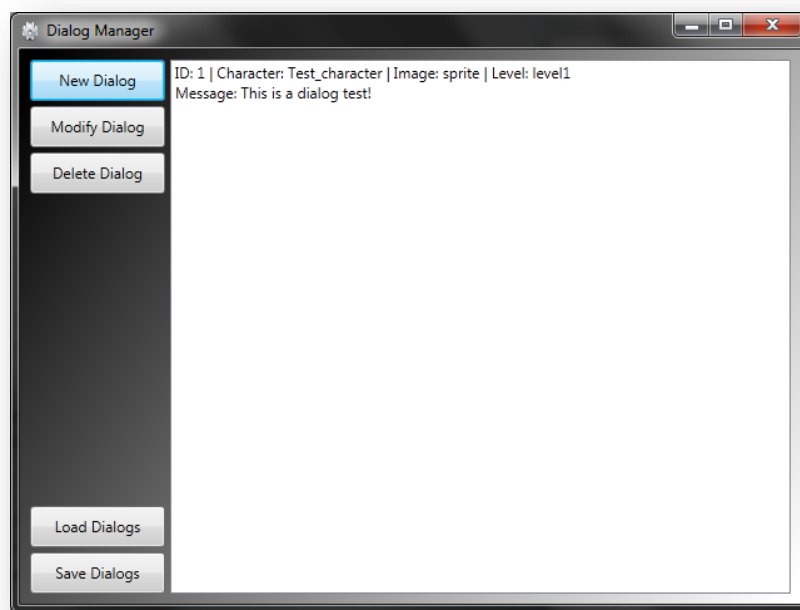


Fig 4. 19 - Screenshot of a new inserted dialog.

To modify an existing dialog, the user just needs to select the desired quest from the list (see figure 4.20A), and click on the “Modify” button. After that, a window pops-up with all the dialog information (see figure 4.20B).

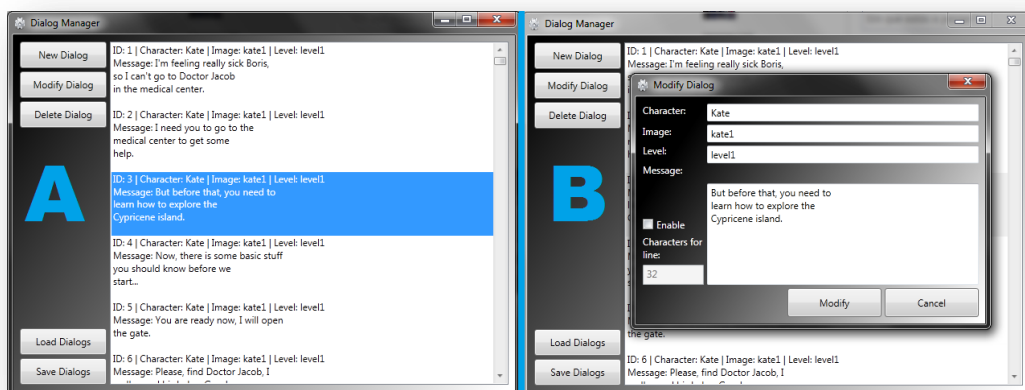


Fig 4. 20 - A: Selecting a dialog from the list; B: Modifying the dialog data.

To load an existing file, the user just needs to click on the “Load Dialogs” button, and indicate the location of the file.

When the user clicks on the “Save Dialogs” button, all information of the list is saved into an XML file that stores all data. The same XML file is later loaded to the game before the level starts, allowing the engine to load all quests on run-time.

The XML file format used for the items is the following:

```
<Dialogs>
  <Dialog>
    <ID> id </ID>
    <Character> character name </Character>
    <Image> character sprite </Image>
    <Level> game level </Level>
    <Message> dialog message </Message>
  </Dialog>
</Dialogs>
```

Note that the character per line information is only used when saving the dialog text information, meaning that the character separation is done every time a dialog is created or modified. Therefore, when the description is saved on the XML file, the character separation is already done.

Once the file is created, the game developer just needs to save the file into the XMLDATA folder of the game content.

4.5. Terrain Creator

This tool differs from the others since it produces 3D assets to be used on the game, namely the terrain for the levels.

Just like the previous tools, the interface of the application (see figure 4.22), is quite simple. It's composed by an area where user can draw the heightmap, and a second area where he can control features like the pen size, the height level, or the textures to be applied on the 3D mesh.

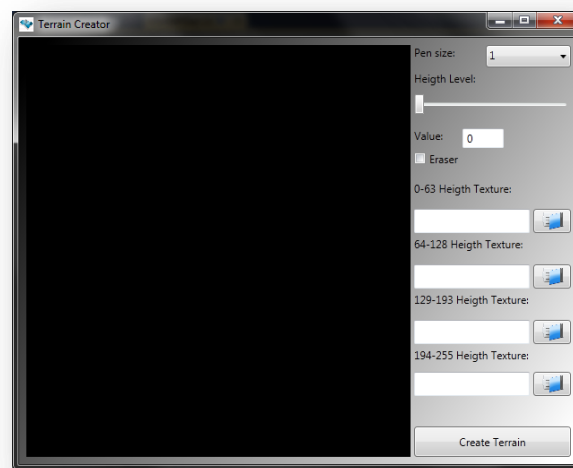


Fig 4. 21 - Screenshot of the Terrain Creator tool.

The Terrain Creator works in a very simple way. The user draws the height areas on the application in levels of grey, these levels go from 0, which is represented by the color black (the lowest of the terrain), to 255 that is the color white (the highest level of the terrain). The user can select the size of the brush by selecting its value from the pen combo box.

When painting the heightmap, the user can change the grey levels in order to create different heights on the terrain. To do that, the user can move the slide bar of the height level, or, if he wants a more controlled change in the height, he can insert the grey value (from 0 to 255) on the height text box (see figure 4.23).

Note that the heights of the generated model will always correspond to the receptive values on the heighmap, meaning that an area with the height value of 30 will have the value 30 on the Z axis.

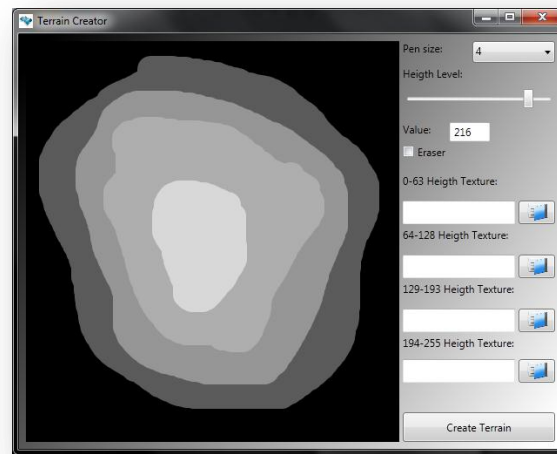


Fig 4. 22 - Painting the heightmap with different pen sizes and height levels.

The user can also erase painted areas by activating the eraser (see figure 4.24). The size of the eraser can also be changed by the pen size combo box.

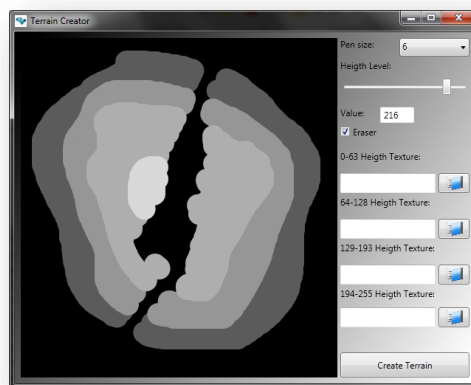


Fig 4. 23 - Erasing the heightmap with the eraser tool.

The textures that will be applied on 3D model representing the terrain can be defined inside the Terrain Creator (see figure 4.25). The application

allows a total of four different textures to be used. The usage of the textures is defined by the height level of the model faces. Four height values are defined in the application for the textures:

- Height between 0 - 63: texture one;
- Height between 64 - 128: texture two;
- Height between 129 - 193: texture three;
- Height between 194 - 255: texture four;

For example, if the vertexes of a face are located on the height level 35, the texture to be applied is the one defined for the interval 0 to 63; this means that the face will have the first texture. In cases where the vertexes are on different levels of height, for example two on level 70 and the other two on level 30, the applied texture is the one of the vertex in the lowest height (i.e., texture of the interval 0 - 63).

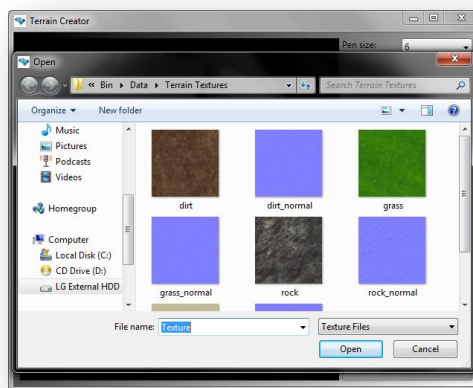


Fig 4. 24 - Loading texture files to the terrain.

Once the map is drawn and the textures selected, the user just needs to press the “Create Terrain” button. The application will then generate an .obj file and an .mtl file with all the information. This model is saved on the Terrain Creator directory with the default name of “terrain”. If the 3D model is correctly generated, a message is showed to the user (see figure 4.26).

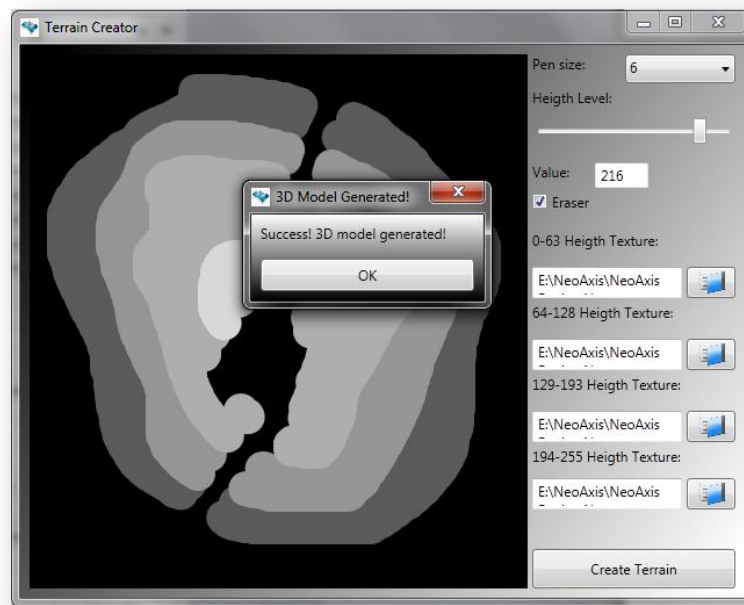


Fig 4. 25 - Terrain successfully created.

The algorithm used for the generation of the 3D model of the terrain is the following:

1. We divide the height and width of the bitmap by 8 in order to simplify the final mesh;
2. We will then visit the pixels of the bitmap, jumping in intervals of 8, saving the x and y coordinates and getting the grey value of the pixel;
3. We create a vertex with the x and y values and the grey level, where x is the value of the x axis, y the value of the y axis, and the grey value the value on the z axis;
4. With the vertex list created, we will then create the faces of the model, by connecting every vertex to their adjacent vertexes;
5. The collected information is then saved into an .obj file;

The created terrain can later be imported to CAD software like 3Ds Max for editing or format conversion (see figure 4.24).

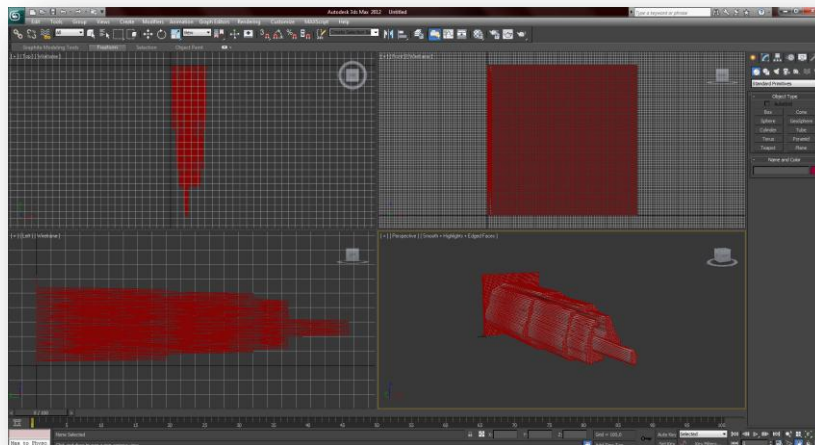


Fig 4. 26 - Created terrain on Autodesk 3Ds Max.

4.6. Connection with Clean World

Three of the applications presented on this Chapter needed some game source code changes in order to work properly; we are talking about the tools that generate XML files to load in the game: Item Manager, Quest Manager, and Dialog Manager.

On the first version of Clean World, all game information regarding items, quests and dialogs was added by code or by sprites, meaning that was static content. Just for dialogs, were loaded into the game 183 different sprites. With the use of the XML files, those sprites are no longer needed, meaning that we can replace the 183 sprites for 1 sprite that will be the text background. The same thing happens with the 32 quests sprites.

The code modifications needed to work with the XML files were quite simple. Once the XML files were on the game content, several modifications had to be done on the level classes. The first thing to do was to create three different lists to store all level information regarding items, quests and dialogs and the respective classes. Once the lists were created,

before any game content like the dialogs was created, all XML information was loaded to the lists where it was stored.

In the case of the dialogs, the only information on XML file that was going to be loaded to the level was the one marked with the current level label. This was done by checking the value of the Level tag on the XML file.

With the XML information loaded into our lists, we proceeded to create the respective items, quests and dialogs for the level we were working on. Several changes had to be done in order to adapt the exiting classes, namely the creation of a new constructor that allows the information to be loaded through the lists when creating the asset. Based on the used constructor, the classes can now create the assets based on a sprite (previous method), or through the information given in the list, using a single sprite for background and the text from the give XML data.

Once these modifications were implemented, the game loaded the content and played smoothly, giving no perception to the player of the modifications done in the source code.

4.7. Summary

On this Chapter were presented the developed tools created with the purpose of optimizing the development process of Clean World game. In Section 4.1 was presented a brief introduction to the developed applications. In Section 4.2 was presented the Item Manager, a tool that allows the user to control all items that can be used in the game. Section 4.3 presented the Quest manager, a tool for the creation and edition of Clean World quests. The Dialog Manager was presented on Section 4.4. This tool allows the creation of the dialogs that are showed during the game. On section 4.5 was presented the Terrain Creator, a tool for the creation of terrains for the game. Finally, on section 4.6 were briefly described the modifications needed do to on Clean World in order to work with the developed applications.

5. Conclusions and Future Work

5.1. Conclusions

Throughout this dissertation we have seen the state-of-the-art of game development tools, the developed game Clean World, and the set of tools created to improve the game development process. This chapter presents the results obtained by the tools optimizations, and points to several directions for future work.

Although XNA is a great tool for beginners on game development, it lacks the tools provided by other engines and frameworks. This flaw makes the game development on this technology harder and slower, since everything must be done by code.

Using the developed tools, creating the game content is now an easier process. The procedure of creating dialogs and quests can now be done through the developed applications, avoiding the creation of unnecessary sprites. The management of the items was also simplified, since the developers can create new items for the game without programming. The use of these applications, not only eliminated around 20 MB of unnecessary sprites, but also the time spent creating the content and introducing it in the game was reduced to half. Furthermore, the developed applications allow the modification of existing content, which means that in case of necessary adjustments, for example a correction on a dialog, the user can alter directly the information without the need to create an entirely new

dialog. This means that all partial objectives presented in Chapter 1 were successfully completed, resulting in the presented dissertation.

5.2. Future Work

To conclude this dissertation, it remains to suggest future research directions that result from this research work:

- The development of a logic editor for the creation of AI for the game characters. Right now, there are no enemies in the created demo of Clean World, however, it would be a feature that we would like to include on a final version of the game. In order to create the enemies, some AI has to be created in order to define the enemies' behaviors. It would be interesting to develop a tool where these behaviors can be created without the need to directly program each enemy. The application could consist on a set of predefined behaviors that could be altered through several parameters, or combined with other existing predefined behaviors.
- The development of an event editor in order to define specific events on the game, for example interacting with a machine or changing levels. On Clean World, every interaction the player does with characters or machines was programmed directly by the programmers, which took a lot of time on the development process. It would be interesting to have an application with a set of predefined interactions; these interactions could be altered through several parameters and called on the game. Thought this system a lot of time would be saved during the level programming.
- The creation of a visual editor for the creation of particle systems to be used in the game. There were no particle systems

implemented on Clean World. It would be a major improvement for the game the existence of tool to create special effects based on particle systems, for example to simulate smoke, water, fire, etc. In order to use particle systems, a particle engine would be added to the physics and render engines. To turn the creation of the particles more intuitive and efficient, could be created a tool for the edition of particle systems, where the user could arrange a set of parameters (for example particle life spawn) in other to obtain the desired result for the specified particle systems. The information regarding the particles could be saved in a XML file that would be loaded into the game. With the information on the XML, the game would be able to recreate the particle system created on the application.

References

- [1] "UDK," [Online]. Available: <http://www.unrealengine.com/udk> [Accessed: August 2012].
- [2] "Epic Games," [Online]. Available: <http://epicgames.com/> [Accessed: August 2012].
- [3] "RPG Maker," [Online]. Available: <http://www.rpgmakerweb.com/> [Accessed: November 2011].
- [4] "Game Maker," [Online]. Available: <http://www.yoyogames.com/gamemaker/windows/> [Accessed: August 2012].
- [5] P. Banaschak, "Early East Asian Chess Pieces: An overview," Issue August 1999.
- [6] B. Neto, L. Fernandes, C. Werner, and J. Moreira de Souza, "Reuse in Digital Game Development," in *Proceedings of the 4th International Conference on Ubiquitous Information Technologies & Applications, 2009. ICUT '09.*, 20-22 Dec. 2009, 2009, pp. 1-6.
- [7] E. S. Association, "Essential facts about the computer and video game industry," *Entertainment Software Association*, Issue 2009.
- [8] D. Callele, E. Neufeld, and K. Schneider, "Requirements engineering and the creative process in the video game industry," in *13th IEEE International Conference on Requirements Engineering, 2005. Proceedings.*, 29 Aug.-2 Sept. 2005, 2005, pp. 240-250.
- [9] A. Brownsword, "Reflecting on development processes in the video game industry," in *ICSE-Companion 2009. 31st International Conference on Software Engineering - Companion Volume, 2009.*, 16-24 May 2009, 2009, pp. 182-182.
- [10] R. L. B. de Barros, C. F. Alves, and G. L. Ramalho, "Investigating the Communication Process in Multidisciplinary Game Development Teams," in *2009 Simposio Brasileiro de Sistemas Colaborativos (SBSC)*, 5-7 Oct. 2009, 2009, pp. 61-69.
- [11] C. M. Kanode and H. M. Haddad, "Software Engineering Challenges in Game Development," in *Sixth International Conference on Information Technology: New Generations, 2009. ITNG '09*, 27-29 April 2009, 2009, pp. 260-265.
- [12] C. Keith, "Get in the Game: What others can learn from game developers," *Better Software Magazine*, Issue November 2006.
- [13] V. T. S. A. L. Apolinário, "A Feature Model Proposal for Computer Games Design," *Proceedings of the VII Brazilian Symposium on Computer Games and Digital Entertainment*, Issue pp. 54-63, 2008.
- [14] A. Furtado and A. Santos, "Applying Domain-Specific Modeling to Game Development with the Microsoft DSL Tools," in *Brazilian Symposium on Computer Games and Digital Entertainment*, 2006.
- [15] V. T. Sarinho, Apolina, x, and A. L. rio, "A Generative Programming Approach for Game Development," in *2009 VIII Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, 8-10 Oct. 2009, 2009, pp. 83-92.

- [16] M. Chen, Y. Zhang, J. Ouyang, and G.-t. Hu, "Game Design and Development Based on Logical Animation Platform," in *2010 International Conference on Computational and Information Sciences (ICCIS)*, 17-19 Dec. 2010, 2010, pp. 573-576.
- [17] W. C. W. Chen, "Game engine applied in education," *Computer and Digital Engineering*, Issue 2008.
- [18] H. T. Y. Song, "Analysis of game engine technology," *Fujian PC*, Issue 2007.
- [19] L. Bishop, D. Eberly, T. Whitted, M. Finch, and M. Shantz, "Designing a PC game engine," *IEEE Computer Graphics and Applications*, vol. 18, Issue 1, pp. 46-53, 1998.
- [20] N. Seung Seok, H. Sung Dea, and P. Jin Wan, "Using a Game Engine Technique to Produce 3D Entertainment Contents," in *16th International Conference on Artificial Reality and Telexistence--Workshops, 2006. ICAT '06*, Nov. 2006, 2006, pp. 246-251.
- [21] "Euphoria," [Online]. Available: <http://www.wtemp.naturalmotion.com/products/euphoria/> [Accessed: August 2012].
- [22] "Yo Yo Games," [Online]. Available: <http://www.yoyogames.com/> [Accessed: November 2011].
- [23] "The 3D Gamemaker," [Online]. Available: http://www.thegamecreators.com/?m=view_product&id=2126 [Accessed: November 2011].
- [24] "The Game Creators," [Online]. Available: <http://www.thegamecreators.com/> [Accessed: November 2011].
- [25] "CopperCube," [Online]. Available: <http://www.ambiera.com/coppercube/> [Accessed: November 2011].
- [26] "Ambiera," [Online]. Available: <http://www.ambiera.com/> [Accessed: November 2011].
- [27] "3D Rad," [Online]. Available: <http://www.3drad.com/> [Accessed: November 2011].
- [28] "Enterbrain," [Online]. Available: <http://www.enterbrain.co.jp/> [Accessed: November 2011].
- [29] "NeoAxis Engine," [Online]. Available: <http://www.neoaxis.com/> [Accessed: August 2012].
- [30] "Unity 3D," [Online]. Available: <http://unity3d.com/> [Accessed: August 2012].
- [31] "Game Maker History," [Online]. Available: http://wiki.yoyogames.com/index.php/Game_Maker_History [Accessed: November 2011].
- [32] "OGRE 3D," [Online]. Available: <http://www.ogre3d.org/> [Accessed: November 2011].
- [33] "DarkBASIC," [Online]. Available: http://www.thegamecreators.com/?m=view_product&id=2030 [Accessed: November 2011].
- [34] "AngelScript," [Online]. Available: <http://www.angelcode.com/angelscript/> [Accessed: November 2011].
- [35] "Big Fish Games," [Online]. Available: <http://www.bigfishgames.com/> [Accessed: November 2011].
- [36] "IGN Entertainment," [Online]. Available: <http://uk.ign.com/> [Accessed: November 2011].
- [37] "GamersGate," [Online]. Available: <http://www.gamersgate.com/> [Accessed: November 2011].
- [38] "Aldorlea Games," [Online]. Available: <http://aldorlea.org/> [Accessed: August 2012].

-
- [39] "*Millennium: A New Hope*," [Online]. Available: <http://aldorlea.org/millennium.php> [Accessed: August 2012].
 - [40] "*Amaranth Games*," [Online]. Available: <http://www.amaranthia.com/> [Accessed: August 2012].
 - [41] "*Aveyond Series*," [Online]. Available: http://www.amaranthia.com/modules/oledrion/product.php?product_id=58 [Accessed: August 2012].
 - [42] "*Blossomsoft*," [Online]. Available: <http://www.blossomsoft.com/> [Accessed: August 2012].
 - [43] "*Eternal Eden*," [Online]. Available: <http://www.blossomsoft.com/?p=182> [Accessed: August 2012].
 - [44] "*Over Cloud 9*," [Online]. Available: <http://www.overcloud9.com/> [Accessed: August 2012].
 - [45] "*Arevan*," [Online]. Available: <http://www.overcloud9.com/games/our-games/arevan/> [Accessed: August 2012].
 - [46] "*Sherman3D*," [Online]. Available: <http://www.sherman3d.com/> [Accessed: August 2012].
 - [47] "*Alpha Kimori*," [Online]. Available: <http://www.alphakimori.com/> [Accessed: August 2012].
 - [48] "*OHBA*," [Online]. Available: <http://ohbado.sakura.ne.jp/> [Accessed: August 2012].
 - [49] "*Homura Combat*," [Online]. Available: <http://ohbado.sakura.ne.jp/homcombat/> [Accessed: August 2012].
 - [50] "*Dream Dale*," [Online]. Available: <http://www.dreamdale.com/about.html> [Accessed: August 2012].
 - [51] "*Elementary My Dear Majesty!*," [Online]. Available: <http://www.alawar.com/game/elementary-my-dear-majesty/> [Accessed: August 2012].
 - [52] "*Makivision Games*," [Online]. Available: <http://www.makivision.com/> [Accessed: August 2012].
 - [53] "*Sacraboar*," [Online]. Available: <http://www.sacraboar.com/> [Accessed: August 2012].
 - [54] "*Donsoft Entertainment*," [Online]. Available: <http://www.donsoft.com.br/> [Accessed: August 2012].
 - [55] "*Capoeira Legends: Path To Freedom*," [Online]. Available: <http://www.capoeiralegends.com> [Accessed: August 2012].
 - [56] "*MisfitVillager*," [Online]. Available: <http://www.indiedb.com/members/sickbrick> [Accessed: August 2012].
 - [57] "*SickBrick*," [Online]. Available: <http://magrathean.ca/sickbrick/> [Accessed: August 2012].
 - [58] "*Magrathean Technologies*," [Online]. Available: <http://magrathean.ca/> [Accessed: August 2012].
 - [59] "*Incognito Episode 3*," [Online]. Available: <http://magrathean.ca/incognito-episode-3/> [Accessed: August 2012].
 - [60] "*Clockwork Brains*," [Online]. Available: <http://clockwork-brains.blogspot.pt/> [Accessed: August 2012].
 - [61] "*Plyushkin Syndrome*," [Online]. Available: <http://clockwork-brains.blogspot.pt/search/label/Plyushkin%20Syndrome> [Accessed: August 2012].
 - [62] "*The Wall Street Journal*," [Online]. Available: <http://online.wsj.com/article/SB10001424052748703904304575497473735761294.html> [Accessed: August 2012].
 - [63] "*NVIDIA® PhysX®*" [Online]. Available: <http://www.geforce.co.uk/hardware/technology/physx> [Accessed: August 2012].
 - [64] "*FMOD*," [Online]. Available: <http://www.fmod.org/> [Accessed: August 2012].
-

References

- [65] "Mono," [Online]. Available: http://mono-project.com/Main_Page [Accessed: August 2012].
- [66] "GameArt Studio GmbH," [Online]. Available: <http://www.gameartstudio.de/> [Accessed: August 2012].
- [67] "A.I. Invasion," [Online]. Available: <http://www.ai-invasion.de/> [Accessed: August 2012].
- [68] "Crescent Moon Games," [Online]. Available: <http://crescentmoongames.com/> [Accessed: August 2012].
- [69] "Aralon: Sword and Shadow," [Online]. Available: <http://www.worldofaralon.com/> [Accessed: August 2012].
- [70] "Bigpoint," [Online]. Available: <http://pt.bigpoint.com/> [Accessed: August 2012].
- [71] "Battlestar Galactica Online," [Online]. Available: <http://pt.battlestar-galactica.bigpoint.com/> [Accessed: August 2012].
- [72] "NPlay," [Online]. Available: <http://www.nplay.com/> [Accessed: August 2012].
- [73] "BeGone," [Online]. Available: <http://www.nplay.com/BeGone/> [Accessed: August 2012].
- [74] "SilverTree Media," [Online]. Available: <http://silvertreemedia.com/animation> [Accessed: August 2012].
- [75] "Cody," [Online]. Available: <http://silvertreemedia.com/products/cordy> [Accessed: August 2012].
- [76] "Limbic Entertainment," [Online]. Available: <http://www.limbic-entertainment.de/> [Accessed: August 2012].
- [77] "Dungeon Empires," [Online]. Available: <http://play.dungeonempires.de/index.php?page=play&lang=2> [Accessed: August 2012].
- [78] "Fun Bits," [Online]. Available: <http://www.funbits.com/> [Accessed: August 2012].
- [79] "Escape Plan," [Online]. Available: <http://uk.playstation.com/escapeplan/> [Accessed: August 2012].
- [80] "Unreal Engine," [Online]. Available: <http://www.unrealengine.com/> [Accessed: November 2011].
- [81] A. Thorn, *UDK Game Development*, 1 ed. vol. 1: Course Technology PTR, 2011.
- [82] "High Moon Studios," [Online]. Available: <http://www.highmoonstudios.com/community/hms> [Accessed: August 2012].
- [83] "Transformers: Fall of Cybertron," [Online]. Available: <http://www.transformersgame.com/> [Accessed: August 2012].
- [84] "Rocksteady Studios," [Online]. Available: <http://www.rocksteadyltd.com/> [Accessed: August 2012].
- [85] "Batman: Arkham City," [Online]. Available: <http://community.batmanarkhamcity.com/> [Accessed: August 2012].
- [86] "BioWare," [Online]. Available: <http://www.bioware.com/> [Accessed: August 2012].
- [87] "Mass Effect 3," [Online]. Available: <http://masseffect.bioware.com/agegate/?url=%2F> [Accessed: August 2012].
- [88] "Gears of War," [Online]. Available: <http://gearsofwar.xbox.com/en-US/AgeGate?source=%252f> [Accessed: August 2012].
- [89] "Grasshopper Manufacture," [Online]. Available: <http://www.grasshoppermanufacture.com/en/index.html> [Accessed: August 2012].
- [90] "Shadows of the Damned," [Online]. Available: <http://www.ea.com/shadows-of-the-damned> [Accessed: August 2012].
- [91] "inXile Entertainment," [Online]. Available: <http://www.inxile-entertainment.com/> [Accessed: August 2012].

-
- [92] "*Hunted: The Demon's Forge*," [Online]. Available: <http://www.huntedthegame.com/index.php/en/index/agegate> [Accessed: August 2012].
- [93] "*2K Games*," [Online]. Available: <http://www.2kgames.com/#/> [Accessed: August 2012].
- [94] "*BioShock 2*," [Online]. Available: <http://www.bioshock2game.com/> [Accessed: August 2012].
- [95] "*SunBurn Engine*," [Online]. Available: <http://www.synapsegaming.com/products/sunburn/engine/> [Accessed: August 2012].
- [96] "*DigitalRune Engine*," [Online]. Available: <http://www.digitalrune.com/> [Accessed: August 2012].
- [97] "*Autodesk 3Ds Max*," [Online]. Available: <http://usa.autodesk.com/3ds-max/> [Accessed: August 2012].
- [98] "*Adobe Photoshop*," [Online]. Available: <http://www.adobe.com/products/photoshopfamily.html?promoid=JOLIW> [Accessed: August 2012].
- [99] "*Panda DirectX Exporter*," [Online]. Available: <http://www.andytather.co.uk/panda/directxmax.aspx> [Accessed: August 2012].
- [100] "*NVIDIA Texture Tools for Adobe Photoshop*," [Online]. Available: <http://developer.nvidia.com/content/nvidia-texture-tools-adobe-photoshop> [Accessed: August 2012].
- [101] "*SunBurn Game Engine*," [Online]. Available: <http://www.synapsegaming.com/products/sunburn/engine/> [Accessed: August 2012].