DEPARTMENT OF COMPUTER SCIENCE LEVEL 400

PROJECT TOPIC

DEVELOPMENT OF A SMART GAMING SOFTWARE FOR BASIC LEVEL STUDENTS IN GHANA

BY:

GEORGE NUAMAH

10124321

JUNE 2012
DECLARATION
I hereby declare that this submission is my own work towards the BSC and that, to the best of my knowledge, it contains no material previously published by another personal nor material which has been accepted for the award of another degree of the University, except where due acknowledgement has been made in the text.

Certified by
Ms. Linda Amoako .................................................. ........................................
(Supervisor) Signature Date

Certified by
Ms. Linda Amoako .................................................. ........................................
(Head of Department) Signature Date
DEDICATION
The entire work is dedicated to my mom Madam Dora Arday, my dad Mr. Alex Afoakwa Nuamah, and my siblings Henry Kojo Nuamah, Alex Nuamah, Henry Nuamah, and Monica Nuamah. I’m grateful for their contribution and moral support in bringing this project to pass.
ACKNOWLEDGEMENTS
The completion of this project marks the end of a very important stage in my academic life that without constructive advice and support of many people would have been impossible to accomplish. Therefore, I would like to take this opportunity to express my appreciation to all of you who had the time and energy to help me throughout this process.
First of all, I would like to thank my supervisor Madam Linda Amoako for her remarkable dedication, useful insights and comments during my work. Moreover, I would like to thank my colleague George Amoako for his helpful ideas whiles testing the software. Last but not least, I would like to take this opportunity to thank my Parents Mr. Alex Afoakwa Nuamah and Madam Dora Ayorkor Arday for their support and prayers.
TABLE OF CONTENTS

DECLARATION ...........................................................................................................i
DEDICATION ..................................................................................................................ii
ACKNOWLEDGEMENTS .................................................................................................. iii
TABLE OF CONTENTS .................................................................................................. iv
LIST OF FIGURES .......................................................................................................... vii
LIST OF ABBREVIATIONS ............................................................................................. viii
CHAPTER ONE .............................................................................................................. Error! Bookmark not defined.
INTRODUCTION ............................................................................................................ Error! Bookmark not defined.
1.1 Background of Design ......................................................................................... Error! Bookmark not defined.
1.2 Problem Statement ............................................................................................... Error! Bookmark not defined.
1.2 Objective of the design .......................................................................................... Error! Bookmark not defined.
1.5 Justification of Design ......................................................................................... 5
1.7 Scope of the Design ............................................................................................... 5
1.8 Limitation of the Design ....................................................................................... 6
11.9 Organization of the Design .................................................................................. 6
1.10 Summary ............................................................................................................... 7
CHAPTER TWO .............................................................................................................. 8
LITERATURE REVIEW ................................................................................................. 8
2.1 Introduction............................................................................................................. 8
2.2 Definition and scope of smart gaming software .................................................. 8
2.3 Categories of SMART video games ...................................................................... 10
2.3.0 Subcategories ..................................................................................................... 10
2.3.1 Brain fitness video games .................................................................................. 10
2.3.2 Children’s educational video games ................................................................. 11
2.3.3 Christian video games ...................................................................................... 11
2.3.4 Criminal law video games ................................................................................ 11
2.3.5 Drawing video games ....................................................................................... 11
2.3.6 Driver training simulators ................................................................................ 12
2.3.6 Educational MUDs............................................................................................ 12
CHAPTER ONE
INTRODUCTION

1.1 Background of Design
According to Mary Bellis (Computer and Video Game History, 1997), in 1952, A.S. Douglas wrote his PhD degree at the University of Cambridge on Human-Computer interaction. Douglas created the first graphical computer game – a version of Tic-Tac-Toe. The game was programmed on an EDSAC (Electronic Delay Storage Automatic Calculator) vacuum-tube computer, which had a cathode ray tube display.

Mary Bellis (Computer and Video Game History, 1997) also wrote that, William Higinbotham created the first video game ever in 1958. His game, called “Tennis for Two,” was created and played on a Brookhaven National Laboratory oscilloscope. In 1962, Steve Russell invented “SpaceWar”. Spacewar was the first game intended for computer use. Russell used a MIT PDP-1 mainframe computer to design his game.

As reported by Mary Bellis (Computer and Video Game History, 1997), in 1967, Ralph Baer wrote the first video game played on a television set, a game called Chase. Ralph Baer was then part of Sanders Associates, a military electronics firm. Ralph Baer first conceived of his idea in 1951 while working for Loral, a television company.

In accordance with Mary Bellis (Computer and Video Game History, 1997), in 1971, Nolan Bushnell together with Ted Dabney, created the first arcade game. It was called Computer Space, based on Steve Russell’s earlier game of Spacewar. The arcade game Pong was created by Nolan Bushnell (with the help of Al Alcorn) a year later in 1972. Nolan Bushnell and Ted Dabney start Atari Computers that same year. In 1975, Atari re-released Pong as a home video game.

As stated in Mary Bellis (Computer and Video Game History, 1997), Larry Kerecman was one of the first operators of video arcade games, including Computer Space. He writes that, “The brilliance of these machines was that Nolan Bushnell and company took what was computer programming (in Space War) and translated it into a simpler version of the game (no gravity) using hard-wired logic circuits. The printed circuit boards that comprise electronics of these games use integrated circuits called small-scale integrated circuits. They consist of discreet logic chips and gates or gates, 4-line to 16-line decoders, etc. straight out of Texas Instruments
catalog. The shape of the rocket ship and flying saucer even are visible in a pattern of diodes on the PC board.”

In 1972 (Bellis, 1997), the first commercial video game console that could be played in the home, The Odyssey, was released by Magnavox and designed by Ralph Baer. The game machine was originally designed while Ralph Baer was still at Sanders Associates in 1966. Baer managed to gain his legal rights to the machine after Sanders Associates rejected it. The Odyssey came programmed with twelve game levels.

In 1976 (Bellis, 1997), Fairchild released the first programmable home game console called the Fairchild Video Entertainment System, and later renamed Channel F. Channel F was one of the first electronic systems to use the newly invented microchip invented by Robert Noyce for the Fairchild Semiconductor Corporation that allowed video games to not be limited by the number of TTL switches (Transistor Transistor Logic Switches, used to both isolate inputs and perform the logic switching).

In consonance with Mary Bellis (Computer and Video Game History, 1997) On June 17, 1980, Atari’s “Asteroids” and “Lunar Lander” were the first two video games to ever be registered in the Copyright Office.

According to Clay Dillow (A Brief History of Video Games, 2011), more than 30 years after Atari popularized the interchangeable game cartridge, gaming-software sales earned an amount of $33 billion annually with the global industry value greater than $100 billion. As players big and small converged at E3 in Los Angeles to catch a glimpse of gaming’s future, we took a look at the innovations that got us here.

As reported by Clay Dillow (A Brief History of Video Games, 2011), the Magnavox Odyssey, 1972, was the world’s first home gaming console which invaded living rooms, and sold 300,000 units. The Atari 2600, 1977, brought variety to the home with its game cartridges. Fan favorites like “Space Invaders” pushed Atari sales past 25 million units over its lifetime. In 1983, Japan was introduced to the Family Computer, or the Nintendo Entertainment System. Two years later, Super Mario Bros. hit U.S. shelves. Mobile gaming went “main-stream” in 1989 with the advent of Nintendo’s Game Boy. Bundled with addictive digital drug Tetris, it went on to sell 118 million units. Engineered to accommodate rapidly improving graphics, Sony’s PlayStation changed gaming in 1994. Released in 2000, PlayStation 2 sold 150 million units. Second Life’s avatar-driven 3-D world had no objectives, winners, or losers, but a massive in-game economy
of 2003: It traded $119 million in virtual goods in 2010. The “World of Warcraft” in 2004 opened its virtual fantasy planet of Azeroth to 12 million paying subscribers. A robust black market peddled in-game goods for real-world cash. In response to Nintendo Wii’s motion-based interface, Microsoft introduced Kinect for Xbox 360 in 2010, which tracked users’ motions, no controller required. At just 3.5 inches, Nintendo’s 3DS in 2011 made small-screen history by introducing the first mainstream glasses-free 3-D display.

1.2 Problem Statement
The intended function of the Smart gaming software is to transform an ordinary video game play into a highly effective treatment for attention deficit, hyperactivity, autistic disorder etc. in basic level students in Ghana. According to Addiction Blog (Top 10 Negative Effects of Games on Children, 2009), many parents worry, no doubt, about the effect video games have on their children. Some of the most common problems recently investigated include, an increase in emotional disorder symptoms, an increase in behavioral disorder symptoms, declines in verbal memory performance, somatic complaints, attention problems such as hyperactivity, detrimental school performance (as video game usage increases, GPA and SAT scores decrease), family interaction problems such as less positive parental relations, significant reduced amounts of sleep, modifications in visual selective attention, playing violent video games is a significant risk factor for later physically aggressive behavior.

In accordance to Kaso Ari (Video Games Side Effects, 2008), what violent video games do to children mentally is very awful. The idea of blowing up buildings, stealing cars or even stabbing someone to death for fun all come from playing violent video games. At a young age as the child start playing these violent games, the bad information about killing people; stealing things, will all be stuck in his/her mind. This wrongly received information transfers into fear, stress and anger causing them to do unimaginable things, which could lead into someone being harmed.

Ingmar Kerem (The Side Effects of Violent Video Games, 2008) stated that, according to research conducted by the American Psychological Association’s (APA) Journal of Personality and Social Psychology (Anderson and Dill, 2000), they concluded that playing violent video games increased a person’s aggressive thoughts, feelings and behavior in both laboratory tests and real life. The (APA) conducted a test where they took 210 college students, made half of
them play a violent video game like “Doom”, while the other half played a non-violent game like “Myst”. After a short time of playing, every student was handed a noise blaster to allow them to blast another person with a loud sound. Results determined that the students that played violent video games blasted noise at another person for a longer period of time on average than those that played the non-violent game. This was a short term test that showed that even a small amount of exposure to violent video games can have an effect on a person’s behavior. Another study by the APA (Anderson and Dill, 2000), conducted in 2000 had 227 college students surveyed about video games. They concluded that those who played video games regularly through middle and high school had a higher aggression level to those that did not. The study also concluded that there was a connection between the amounts of time spent playing games through middle and high school to decreasing grades in college.

(Craig A. Anderson and Karen E. Dill, 2000)

1.3 Objective of the design
The main objective of this project is to design a video game that would create a platform for children to be able to play and learn at the same time whiles taking out violence and any other negative feature from the picture. The following objectives can be identified as a result:

i. Identify good and bad behavioral features associated with the gaming software for Ghanaian Students so that negative effects of bad behaviors as a result violent video games may be controlled.

ii. Categorizing the behaviors and mannerisms into appropriate and inappropriate with the gaming software for students in Ghana. This will help decipher which of these features can be incorporated in the design to harness the good morals in the children.

iii. Assessing the impact of behavioral traits inculcated into the game design for basic level students in Ghana in order to bring out the best character traits in our future leaders.

iv. Identifying some of the methods employed by game developers in controlling the behavior traits given to the gaming software in order to adopt them in the game design.

1.5 Justification of Design
With reference to the literature, it is quiet conspicuous that not so much has been done in the area of creating violent-free gaming software. It is unfortunate since the youth of Ghana are
constantly being exposed to these foreign violent infested games such as Grand Theft and Doom leaving no room for healthier games. This design seeks to contribute to providing our future leaders with revolutionary gaming software pointing them in the right direction.

While this design contributes to other existing designs in the field of gaming it also inspires other programmers to come out with more creative ideas in the software programming industry that can benefit society and the nation as a whole. To this effect the outcome realized as a result of this project will be of immense benefit to stakeholders in the industry. Furthermore, the design will serve as a foundation for future developers who might be interested in taking the project a step further.

1.6 Scope of the Design
The design would be tailored to incorporate the educational and moral features that would be beneficial to both schools and the household. The design would make use of primary and secondary data. The primary data would be gathered from tutors in the classroom and parents in the household. The primary data would be help find out what educational and moral material would be most appropriate to be used in the design of the gaming software. The rest of the information would be gathered from other sources including the internet sources, existing video games, and manuals which contain the latest technology for an up to date modern design.

1.8 Limitations of the design
The major limitation in this design is the limited time frame for the programming. Evidently this is onerous and hence a real challenge. Limited resources to carry out the research and design in the country are another setback. Both software and hardware tools needed for the design of the gaming software are not easily available in the country. They usually cost so much money to obtain from outside the country. Another significant limitation is the access restriction to sensitive information from certain secure private networks on the internet. It is either a restricted area or it requires some form of payment to grant the access. The complex nature of the project can be another potential limitation. It is both tedious to collect information for the design and also to program the software from scratch. It might even take several months or even over a year to actually come out with suitable well-designed software which is bug-free and stable at all times.
1.8 Methodology for the Design
The gaming software would be designed as a means to create an outlet for both children and adults. The test case data will be Adventist Primary and JHS, Educational Syllabus from Ghana Education Service (GES) and Individual Parents and Pupils.

1.9 Organization of the design
This design project would be structured in five (5) chapters. Chapter one, will entitle, introduction to the design, will provide the background information on the design. It will include a statement of the problem, objectives of the design, research questions and significance of the design. Chapter two on the other hand will contain a review of available literature on the area of study. Chapter three will present the methodology employed in the design and detailed information from the primary and secondary sources. Chapter four will capture detailed analysis and its interpretation. Chapter five will detail out the conclusions and give recommendations for improvement and future work.
CHAPTER TWO
LITERATURE REVIEW

2.1 Definition and scope of smart gaming software

The term “Smart” according to Farlex Inc. (Dictionary/Thesaurus, 2012) can be defined as clear and quick in thought and action. It can also refer to briskness or sharpness. Educational games are games that have been designed to teach people about a certain subject, expand concepts, reinforce development, understand a historical event or culture, or assist them in learning a skill as they play. They include board, card, and video games.

Educational video games are considered a type of serious game, as these games have an academic purpose other than pure entertainment. According to Wikipedia (Educational Entertainment, 2012) these types of games are called edutainment because they combine education and entertainment. An educational computer game can be defined as an electronic medium with all the characteristics of a gaming environment that have intended educational outcome targeted at particular groups of learners. Video games can aid the development of proficiency in logical reasoning by allowing users to interact with objects and manipulate variables. According to Mediatec Publishing (Games Create Learning Through Competition, 2012) they are said to be particularly effective when designed to address a specific problem or teach a certain skill in curriculum subjects, where specific objectives can be stated and when deployed selectively within a context relevant to the learning activity and goal. Simple types of games can be designed to address specific learning outcomes such as recall of factual content. For instance, the Nobel Prize Foundation website uses on-line games to aid children in understanding the discoveries made by its laureates by embedding the scientific knowledge as part of the game environment.

According to Cryer (Educational Game Explained, 2009), to aid in educating students and adults about the finer details of different political systems, numerous companies have developed simulations that immerse the player into different political systems by forcing them to make realistic political decisions. These games vary from running an actual election campaign to games that allow the player to make the day-to-day decisions of running a country, as seen in Democracy. These types of games are targeted at students, educators and adults alike.
Cryer (Educational Game Explained, 2009) further explained that, video games have historically received more criticism than other forms of recreational learning because they are often perceived as being mindless entertainment which encourages sexism and consumerism, and turn players into social recluses. Many children today also find these “educational” games to lack interesting content, as they are considered by older children to be “for preschoolers”. However, a shift from pure entertainment to educational tool has emerged in recent times. Children growing up today can benefit from educational video games because they are already exposed to a society that is increasingly dependent on digital technology.

As reported by Cryer (Educational Game Explained, 2009), researchers today have found that computer games could become part of the school curriculum after researchers found they had significant educational value. “The UK study concluded that simulation and adventure games – such as Sim City and RollerCoaster Tycoon, where players create societies or build theme parks, developed children’s strategic thinking and planning skills”. Recent games such as the math game Ko’s Journey are standards-based to meet curriculum.

The medium of educational games provides an opportunity for teachers to introduce educational and playful elements into the learning environment. With computer-aided learning programs, teachers may assist students on social aspects such as critical learning, knowledge based communication and effective interpersonal skills that traditional methods of teaching cannot offer. As computer games are being adapted to the education system, the issue of classification and content regulation is being brought to attention. The issue of regulating game content is vital as educational games are created to be effective learning tools. Thus the game developer must have a comprehensive understanding of its young audience and their particular social and educational needs. At the same time, the game developer must balance between entertainment and the education syllabus.

According to Dostal J (Educational software and computer games, 2009) most of these types games target young users from the ages of about three years to mid-teens. Beyond the mid-teens, subjects become so complex (e.g. calculus) that teaching via a game can be impractical. Numerous subgenres exist, each for a different field, such as math games or typing games. Adult education games are aimed at higher levels of education being targeted at young-adults and up. Like children’s games, they can be in the form of mini-games, adventure games, role-playing games and so on. There can also be defined a strategy for war games that include historical
references like the Total War franchise or the Age of Empires trilogy and an in-game encyclopedia like Civilization. These games often combine entertainment and education but without being explicitly educational.

2.3 Categories of SMART video games
SMART or educational games attempt to teach the users using the game as a vehicle. Educational games are a great tool for building foundation math and language skills that today’s elementary school curriculum requires.

2.3.0 Subcategories
There are vast inclinations of developing an educational game. For the purpose of our design, the focus will be on the academic aspect of educational software. Some subcategories identified under this category include:

2.3.1 Brain fitness video games
According to Ball Birch (American Medical Association, 2002), the Brain Fitness Game Program games are training game software developed as a possible means for improving auditory processing and memory. It approaches this goal by adherence to the principles of brain plasticity. Unlike alternative treatments which propose specific compensation methods as a solution for age-related memory loss, this software attempts to take advantage of natural brain plasticity in an attempt to improve auditory processing and memory. Due to the propensity of the brain to relearn and alter itself based on selective attention to stimuli, the Brain Fitness game software was designed with the goal that the brain would actually be restructured during its use. According to Jaeggi, S., Buschkuehl (Improving fluid intelligence with training on working memory, 2008) The Brain Fitness Program, a computerized brain training program created by Posit Science, was shown to significantly improve memory, attention, and information processing. Those who trained with the software were twice as fast in processing information, and they scored as well on memory and attention tests as those 10 years younger. It was concluded that the experimental program improved generalized measures of memory and attention more than an active control program.
2.3.2 Children’s educational video games
These are educational video games intended for children between the ages of 3 and 16. While most of these games have an Early Childhood (EC) rating according to the Entertainment Software Rating Board (ESRB), some of these games have a K-A/E (Everyone) rating. K-A rating are considered to be E-rated which means everyone over the age of 6 years. Examples of these games include: Carmen Sandiego, The ClueFinders, JumpStart, Keisan Game series, The Magic School Bus Video games, Mia series, Nick Jr. video games, Sesame Street games, Thomas and Friends video games, Video games based on PBS Kids shows and so on.

2.3.3 Christian video games
Christian video games are educational video games that teach religious and moral values. Some examples include Adam’s Venture, Bible 1, Bible Adventures, Bible Adventures, Dance Praise, and King of Kings, Exodus, Noah’s Ark, Sunday Funday, Left Behind: Eternal Forces and Guitar Praise.

2.3.4 Criminal law video games
This category has only one subcategory, Ace Attorney. Some examples include: Harvey Birdman: Attorney at Law, Phoenix Wright: Ace Attorney, Gyakuten Kenji 2, Phoenix Wright: Ace Attorney: Trials and Tribulations, and Professor Layton vs. Ace Attorney.

2.3.5 Drawing video games
As the name signifies, drawing video games teaches the art of drawing. Some examples include: Acme Animation Factory and Art Alive, Chibi Maruko-chan: Mezese Minami no Island, Coloring Book software, Mario Paint, Paint By DS, and Wacky Worlds Creativity Studio.

2.3.5 Driver training simulators
Driver training simulators are useful acquiring driving skills. Examples include: Bus Driver, CarSim, City Bus Simulator, Driving simulator, Drivotrainer, National Advanced Driving Simulator, SimuRide, Urban Jungle, Virtual training and VSTEP (a leading European developer of simulators and virtual training software)
2.3.6 Educational MUD (Multi-User Dimension)
MUD (Multi-User Dimension) refers to a type of multi-user online virtual environment. The following 8 sub-categories of this category include the following: MOO, BioMOO, Costello (online game), Diversity University, LinguaMOO, MediaMOO, MicroMUSE, and MOOSE Crossing.

2.3.7 Environmental education video games
These computer and video games base their gameplay on simulating biological aspects of life, such as survival, genetics or ecosystems. This category has only one subcategory, Biological simulation video games. Examples of these games include: Climate Challenge, Fate of the World, and Games for Change, Global warming game, Stabilization Wedge Game, and V GAS.

2.3.8 Free open source educational games
Free open source educational game software is game software that can be used, studied and modified without restriction, and which can be copied and redistributed either without restriction, or with restriction that only ensure that further recipients can also do these things with the source code made available. Examples include: Food Force 2, GCompris, PlayPower, Tux Paint, Tux Typing, Tux of Math Command, TuxMathScrabble, TuxWordSmith, and Word Whiz.

2.3.9 Geography educational video games
Geography educational video games possess geographical facts and features. Examples include: Globetrotter 2 and La Terre (video game). According to Luke Smith (Globetrotter 2, 2004), Globetrotter 2 is an educational game about geography by Deadline Games. The player has to answer geographical questions in order to progress through the game and move to other countries and cities. La Terre is an educational video game released in 2003, developed and published by Microids. It is designed to allow the user to discover and understand the planet Earth.

2.3.10 History educational video games
History educational video games teach facts about history. Examples include: Dinosaur World (video game), Genesys (video game), Grendel’s Cave, Impressionists (video game), Making

2.3.11 Human-based computation games
A human-based computation game or Gwap (a game with a purpose) is a human based computation technique in which a computational process performs its function by outsourcing certain steps of humans in an entertaining way. This approach uses differences in abilities and alternative costs between humans and computer agents to achieve symbiotic human-computer interaction. These tasks can include labeling images to improve web searching, transcription of ancient text (where Optical Character Recognition software faces a script they are not optimized for and degraded or damaged images) and any activity requiring common sense or human experience. Examples of Gwap (a game with a purpose) include the following: ESP game, EteRNA, Foldit, Human-based computation game, Ontogame, Page Hunt, and Phetch, Phylo (video game).

2.3.12 Humanitarian video games
The Humanitarian video games are games that inculcate humanitarian values into the software. Examples include: Against All Odds (video game), Amnesty the game, Darfur is Dying, FloodSim, Food Force, Food Force 2, Foreign Ground, Freedom, Games for Change, Global Conflict: Palestine, Global conflicts: latin America, Newsgame, Pax Warrior, PeaceMaker (video game), Real Liver, and SimHealth.

2.3.13 Language learning video games
2.3.14 **Mathematical education video games**

2.3.15 **Programming games**
Programming games allow users to study programming codes and instructions. Examples include: Progamming game, BASIC Progamming, Carnage Heart, ChipWits, Colobot, Color Robot Battle, Core War, Crobots, Darwin (programming game), MicroMUSE, MindRover, MOOSE Crossing, Omega (video game), RARS, Robocode, Roboforge, Robot Battle, Robot Battle (Macintosh game), Robot Odyssey, RobotWar, RoboWar, Rocky’s Boots, Stagecast, Creator, TankAI, ToonTalk, and TORCS.

2.3.16 **Quiz video games**
Quiz video games helps in broadening the knowledge base of the user. Examples include: List of quiz arcade games, 1 vs. 100 (Nintendo DS video game), 1 vs. 100 (Xbox 360 video game), Alan Hansen’s Sports Challenge, Backpacker (series), Bakuretsu Quiz Ma-Q Dai Boken, Big Bucks Trivia, Blufr, Buzz, Buzz Junior: Dino Den, Buzz Junior: Jungle Party, Buzz Junior: Monster Rumble, Buzz Junior: Robo Jam, Buzz: and Brain of the World

2.3.17 **Science educational video games**
Science educational video games project scientific knowledge through them. Examples include: Astronomica: The Quest for the Edge of the Universe, Buzz Aldrin’s Race Into Space, Critical Mass (video game), Genomics Digital Lab, History of Biology (video game), Immune Attack, KAtomic, Launchball, MoonBaseOne, NanoMission, Odell Down Under, Quarky & Quaysoo’s Turbo Science and Widget Workshop.
2.3.18 Sex education video games

Sex education video games are designed to teach facts about reproduction. An example is Catch the Sperm. Catch the Sperm is a Swiss computer game recognized as an entertaining way for health professionals to promote prevention of HIV. According to H M Urfer (Catch the Sperm: computer game on the internet for AIDS prevention, 2012) The original Catch the Sperm (now known as CTS Style I) is a 3.4-megabite action game created in 2001 and updated in 2002 and 2003 by Phenomedia for the Swiss Federal Office of Public Health’s STOP AIDS Campaigne. The game can be played on a personal computer, a mobile telephone, or a cellular telephone. The game was designed for free international distribution. Copyright of the Catch The Sperm series is owned by Scandinavian Games.

2.3.19 Typing video games

Typing video games helps in developing the player’s typing skills. Examples include: Battle & Get Pokemon Typing DS, JumpStart Typing, Mavis Beacon Teaches Typing, Typer Shark, TypeRacer, The Typing of the Dead and The Typing of the Dead 2.

2.4 Design concepts

The design concepts provide the software designer with a foundation from which more sophisticated methods can be applied. A set of fundamental design concepts has evolved over the years. They are:

- Abstraction - Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose.
- Refinement - It is the process of elaboration. A hierarchy is developed by decomposing a macroscopic statement of function in a stepwise fashion until programming language statements are reached. In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary.
- Modularity - Software architecture is divided into components called modules.
• Software Architecture - It refers to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system.

• Control Hierarchy - A program structure that represents the organization of a program component and implies a hierarchy of control.

• Structural Partitioning - The program structure can be divided both horizontally and vertically. Horizontal partitions define separate branches of modular hierarchy for each major program function. Vertical partitioning suggests that control and work should be distributed top down in the program structure.

• Data Structure - It is a representation of the logical relationship among individual elements of data.

• Software Procedure - It focuses on the processing of each module individually

• Information Hiding - Modules should be specified and designed so that information contained within a module is inaccessible to other modules that have no need for such information.

2.4.1 Design considerations

There are many aspects to consider in the design of a piece of software. The importance of each should reflect the goals the software is trying to achieve. Some of these aspects are:

• **Compatibility** - The software should be able to operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself. A program is backward compatible if it can use files from an older version of itself. For a file saved in the program to be backward compatible, it must be possible to open the file in a previous version of the program.

• **Extensibility** – It should be possible to add new capabilities to the software without major changes to the underlying architecture. The resulting software comprises well defined, independent components. That leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.
- **Fault-tolerance** – If it is resistant (absolutely) then there is no need for recovery from component failure.

- **Maintainability** – The maintainability index is calculated with certain formulae from lines-of-code measures, McCabe measures and Halstead complexity measures. The measurement and track maintainability are intended to help reduce or reverse a system’s tendency toward “code entropy” or degraded integrity, and to indicate when it becomes cheaper and/or less risky to rewrite the code than it is to change it.

- **Packaging** - Printed material such as the box and manuals should match the style designated for the target market.

- **Reliability** – Accessing reliability requires series of checks including application architecture practices, coding practices, complexity of algorithms and complexity of programming practices.

- **Reusability** – The design features of a software element should be able to enhance its suitability for reuse.

- **Robustness** - The software should be able to operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with resilience to low memory conditions.

- **Security** - The software should be able to withstand hostile acts and influences by placing restrictions with user access.

- **Usability** - The software user interface must be usable for its target user/audience. Default values for the parameters must be chosen so that they are a good choice for the majority of the users. Usability has to do with the ease of use of the software.

### 2.4.2 Modeling language
According to Xiao He (A metamodel for the notation of graphical modeling languages, 2007), a modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure. A modeling language can be graphical or textual. Examples of graphical modeling languages for software design are:
- Business Process Modeling Notation (BPMN) is an example of a Process Modeling language.
- EXPRESS and EXPRESS-G (ISO 10303-11). This is an international standard general-purpose data modeling language.
- Extended Enterprise Modeling Language (EEML). This is a commonly used language for business process modeling across a number of layers.
- Flowchart is a schematic representation of an algorithm or a stepwise process.
- Fundamental Modeling Concepts (FMC) modeling language for software-intensive systems.
- Integration Definition (IDEF) is a family of modeling languages, the most notable of which include Integration Definition for Function Modeling (IDEF0) for functional modeling, Integration Definition for Function Modeling (IDEF1X) for information modeling and Integrated Definition for Ontology Description Capture Method (IDEF5) for modeling ontologies.
- Jackson Structured Programming (JSP) is a method for structured programming based on correspondences between data stream structure and program structure.
- LePUS3 is an object-oriented visual Design Description Language and a formal specification language that is suitable primarily for modeling large object-oriented (Java, C++, C#) programs and design patterns.
- Unified Modeling Language (UML) is a general modeling language to describe software both structurally and behaviorally. It has a graphical notation and allows for extension with a Profile.
- Alloy is a general purpose specification language for expressing complex structural constraints and behavior in a software system. It provides a concise language based on first-order relational logic.
- Systems Modeling Language (SysML) is a new general-purpose modeling language for systems engineering.
2.4.3 Design patterns
A software designer or architect may identify a design problem which has been solved by others before. A template or pattern describing a solution to a common problem is known as a design pattern. The reuse of such patterns can speed up the software development process, having been tested and proven in the past.

2.4.4 Usage
Software design documentation may be reviewed or presented to allow constraints, specifications and even requirements to be adjusted prior to programming. Redesign may occur after review of a programmed simulation or prototype. It is possible to design software in the process of programming, without a plan or requirement analysis, but for more complex projects this would not be considered a professional approach. A separate design prior to programming allows for multidisciplinary designers and Subject Matter Experts (SMEs) to collaborate with highly-skilled programmers for software that is both useful and technically sound.

2.4.5 Coding
Computer programming (often shortened to programming or coding) is the process of designing, writing, testing, debugging, and maintaining the source code of computer programs. This source code is written in one or more programming languages. The purpose of programming is to create a set of instructions that computers use to perform specific operations or to exhibit desired behaviors. The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.

2.4.6 Readability of source code
In computer programming, readability refers to the ease with which a human reader can comprehend the purpose, control flow, and operation of source code. It affects the aspects of quality above, including portability, usability and most importantly maintainability. Readability is important because programmers spend the majority of their time reading, trying to understand and modifying existing source code, rather than writing new source code. Unreadable code often leads to bugs, inefficiencies, and duplicated code. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it.
Following a consistent programming style often helps readability. However, readability is more than just programming style. Many factors have little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Some of these factors include:

- Different indentation styles (whitespace)
- Comments
- Decomposition
- Naming convention for objects (such as variables, classes, procedures, etc.)

2.4.7 Algorithmic complexity
The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problem. For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances.

2.4.8 Measuring language usage
It is very difficult to determine what are the most popular of modern programming languages. Some languages are very popular for particular kinds of applications (e.g., COBOL is still strong in the corporate data center, often on large mainframes, FORTRAN in engineering applications, scripting languages in web development, and C in embedded applications), while some languages are regularly used to write many different kinds of applications. Also many applications use a mix of several languages in their construction and use. New languages are generally designed around the syntax of a previous language with new functionality added (for example C++ adds object-orientedness to C, and Java adds memory management and byte code to C++).

Methods of measuring programming language popularity include: counting the number of job advertisements that mention the language, the number of books teaching the language that are
sold (this overestimates the importance of newer languages), and estimates of the number of existing lines of code written in the language (this underestimates the number of users of business languages such as COBOL).

### 2.4.9 Debugging

Debugging is a very important task in the software development process, because an incorrect program can have significant consequences for its users. Some languages are more prone to some kinds of faults because their specification does not require compilers to perform as much checking as other languages. Use of a static code analysis tool can help detect some possible problems. Debugging is often done with IDEs like Eclipse, Kdevelop, NetBeans, Code::Blocks, and Visual Studio. Standalone debuggers like gdb are also used, and these often provide less of a visual environment, usually using a command line.

### 2.4.10 Programming languages

Different programming languages support different styles of programming (called *programming paradigms*). The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Ideally, the programming language best suited for the task at hand will be selected. Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute. Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly. It is usually easier to code in "high-level" languages than in "low-level" ones.

The details look different in different languages, but a few basic instructions appear in just about every language:

- **input**: Get data from the keyboard, a file, or some other device.
- **output**: Display data on the screen or send data to a file or other device.
- **arithmetic**: Perform basic arithmetical operations like addition and multiplication.
• **conditional execution**: Check for certain conditions and execute the appropriate sequence of statements.

• **repetition**: Perform some action repeatedly, usually with some variation.

Many computer languages provide a mechanism to call functions provided by libraries such as in a `.so`. Provided the functions in a library follow the appropriate run time conventions (e.g., method of passing arguments), then these functions may be written in any other language.

### 2.5 Unit Testing

Unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application.

Unit tests are created by programmers or occasionally by white box testers during the development process. Ideally, each test case is independent from the others: substitutes like method stubs, mock objects, fakes and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended. Its implementation can vary from being very manual (pencil and paper) to being formalized as part of build automation.

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

#### 2.5.1 Find problems early

Unit tests find problems early in the development cycle.

In Test-Driven Development (TDD), which is frequently used in both Extreme Programming and Scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequently as the larger
code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it considered to be a bug either in the changed code or the tests themselves. They unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development team of the problem before handing the code off to testers or clients, it is still early in the development process.

2.5.2 Facilitates change

Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.

Readily available unit tests make it easy for the programmer to check whether a piece of code is still working properly.

In continuous unit testing environments, through the inherent practice of sustained maintenance, unit tests will continue to accurately reflect the intended use of the executable and code in the face of any change. Depending upon established development practices and unit test coverage, up-to-the-second accuracy can be maintained.

2.5.3 Simplifies integration

Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

An elaborate hierarchy of unit tests does not equal integration testing. Integration with peripheral units should be included in integration tests, but not in unit tests. Integration testing typically still relies heavily on humans testing manually; high-level or global-scope testing can be difficult to automate, such that manual testing often appears faster and cheaper.

2.5.4 Documentation

Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit’s API.
Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

By contrast, ordinary narrative documentation is more susceptible to drifting from the implementation of the program and will thus become outdated (e.g., design changes, feature creep, relaxed practices in keeping documents up-to-date).

### 2.5.5 Design
When software is developed using a test-driven approach, the unit test may take the place of formal design. Each unit test can be seen as a design element specifying classes, methods, and observable behavior.

### 2.5.6 Acceptance Testing
Testing generally involves running a suite of tests on the completed system. Each individual test, known as a case, exercises a particular operating condition of the user's environment or feature of the system, and will result in a pass or fail, or Boolean, outcome. There is generally no degree of success or failure. The test environment is usually designed to be identical, or as close as possible, to the anticipated user's environment, including extremes of such. These test cases must each be accompanied by test case input data or a formal description of the operational activities (or both) to be performed—intended to thoroughly exercise the specific case—and a formal description of the expected results.

Acceptance Tests/Criteria (in Agile Software Development) are usually created by business customers and expressed in a business domain language. These are high-level tests to test the completeness of a user story or stories 'played' during any sprint/iteration. These tests are created ideally through collaboration between business customers, business analysts, testers and developers, however the business customers (product owners) are the primary owners of these tests. As the user stories pass their acceptance criteria, the business owners can be sure of the fact that the developers are progressing in the right direction about how the application was
envisaged to work and so it's essential that these tests include both business logic tests as well as UI validation elements (if need be).

Acceptance test cards are ideally created during sprint planning or iteration planning meeting, before development begins so that the developers have a clear idea of what to develop. Sometimes (due to bad planning!) acceptance tests may span multiple stories (that are not implemented in the same sprint) and there are different ways to test them out during actual sprints. One popular technique is to mock external interfaces or data to mimic other stories which might not be played out during an iteration (as those stories may have been relatively lower business priority). A user story is not considered complete until the acceptance tests have passed.

3.5.1 Process

The acceptance test suite is run against the supplied input data or using an acceptance test script to direct the testers. Then the results obtained are compared with the expected results. If there is a correct match for every case, the test suite is said to pass. If not, the system may either be rejected or accepted on conditions previously agreed between the sponsor and the manufacturer.

The objective is to provide confidence that the delivered system meets the business requirements of both sponsors and users. The acceptance phase may also act as the final quality gateway, where any quality defects not previously detected may be uncovered.

A principal purpose of acceptance testing is that, once completed successfully, and provided certain additional (contractually agreed) acceptance criteria are met, the sponsors will then sign off on the system as satisfying the contract (previously agreed between sponsor and manufacturer), and deliver final payment.
CHAPTER THREE
METHODOLOGY

3.0 Introduction
This chapter covers the approach and techniques applied in this project. The techniques would also be described and this would be followed by the strategy, design approach, the techniques and data collection method.

3.1 Requirements Analysis
It encompasses those tasks that go into determining the needs or conditions to be met for a new or altered product, taking into account the possibility of conflicting requirements of the various stakeholders.

Requirements analysis is critical to the success of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

Conceptually, requirements analysis includes three types of activities:

- Eliciting requirements: the task of identifying the various types of requirements from various sources including project documentation, (e.g. the project charter or definition), business process documentation, and stakeholder interviews. This is sometimes also called requirements gathering.
- Analyzing requirements: determining whether the stated requirements are clear, complete, consistent and unambiguous, and resolving any apparent conflicts.
- Recording requirements: Requirements may be documented in various forms, usually including a summary list and may include natural-language documents, use cases, user stories, or process specifications.

Requirements analysis can be a long and arduous process during which many delicate psychological skills are involved. New systems change the environment and relationships between people, so it is important to identify all the stakeholders, take into account all their needs and ensure they understand the implications of the new systems. Analysts can employ
several techniques to elicit the requirements from the customer. These may include the development of scenarios (represented as user stories in agile methods), the identification of use cases, the use of workplace observation or ethnography, holding interviews, or focus groups (more aptly named in this context as requirements workshops, or requirements review sessions) and creating requirements lists. Prototyping may be used to develop an example system that can be demonstrated to stakeholders. Where necessary, the analyst will employ a combination of these methods to establish the exact requirements of the stakeholders, so that a system that meets the business needs is produced.

3.2 Design
Software design is a process of problem solving and planning for a software solution. After the purpose and specifications of software are determined, software developers will design or employ designers to develop a plan for a solution. It includes low-level component and algorithm implementation issues as well as the architectural view.

A software development process, also known as a software development life cycle (SDLC), is a structure imposed on the development of a software product. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

The activities of the software development process represented in the waterfall model

![Diagram of the software development process](image)
As represented in the figure above, the activities of the software development process are Requirements, Design, Implementation, Verification and Maintenance.

The Software Requirements Analysis (SRA) step of a software development process yields specifications that are used in software engineering. If the software is "semiautomated" or user centered, software design may involve user experience to help determine those specifications. If the software is completely automated (meaning no user or user interface), a software design may be as simple as a flow chart or text describing a planned sequence of events. There are also semi-standard methods like Unified Modeling Language (UML) and Fundamental modeling concepts. In either case some documentation of the plan is usually the product of the design.

A software design may be platform-independent or platform-specific, depending on the availability of the technology called for by the design.

**Basic Design:** If the first phase gets successfully completed and a well thought out plan for the software development has been laid then the next step involves formulating the basic design of the software on paper.
CHAPTER FOUR
SMART GAME 1.0 IMPLEMENTATION TESTING AND EVALUATION

4.1 INTRODUCTION
The name “SMART GAME 1.0” was derived from the purpose or intended use of the game. The term smart according to Wikipedia – Dictionary.com (2012) may refer to having or showing a quick-witted intelligence (The ability to acquire and apply knowledge and skills). A game on the other hand may refer to a form of play or sport, especially a competitive one played according to rules and decided by skill, strength or luck. We can therefore conclude that SMART GAME 1.0 is a combination of intelligence and play to achieve skill.

The SMART GAME 1.0 was designed using Adobe Flash Professional CS5 compiler software. The game is made up of twenty-four frames with four (4) layers in each frame.

4.2 FRAMES AND KEYFRAMES
Like films, Adobe Flash Professional CS5 documents divide lengths of time into frames. In the Timeline, these frames are worked with to organize and control the content of your document. Frames are placed in the Timeline in the order preferred for the objects in the frames to appear in the finished content.

A keyframe is a frame where a new symbol instance appears in the Timeline. A keyframe can also be a frame that includes ActionScript code to control some aspect of your document. A
blank keyframe can also be added to the Timeline as a placeholder for symbols which are planned to add later or to explicitly leave the frame blank.

A property keyframe is a frame in which a change to an object’s properties for an animation is defined. Flash Pro can tween, or automatically fill in, the property values between the property keyframes in order to produce fluid animations. Because property keyframes gives room for the production of animation without drawing each individual frame, they make creating animation easier. A series of frames containing tweened animation is called a motion tween.

A tweened frame is any frame that is part of a motion tween. A static frame is any frame that is not part of a motion tween. Keyframes and property keyframes are arranged in the Timeline to control the sequence of events in your document and its animation.

The figure below shows the selected first game frame out of a total of twenty-four (24) frames

Frame 1 of 24
4.3 Drawing Shapes and Objects

The shapes and objects are created using a simple drawing tool found within the Flash programme. The shapes used in the designing of the SMART GAME 1.0 are ovals, rectangles and squares. A pencil tool was also used in making drawings within the programme frames. Other tools used include paint with the Brush tool, drawing with the Pen tool, Apply the Symmetry Brush effect, Apply the Grid Fill effect, etc. These shapes were used to create the buttons, arrow drawings, the ground platform, the ball character, the portal and the enemy.

Below is a pictorial description of samples of each character and shape as used in the programme:

![Figure 4.3](image1.png)

These shapes and objects that are created have to be categorized into two major sections name a Movie Clip and a Button. The button operates on click commands whiles the movie clip operates on motion commands.
4.4 Text
A lot goes into coming out with the most appealing text format for each description in each frame of the game. Care must be taken in order not to use boring character texts which might reduce the level of interest in the game. Some of the character types used in the designing of the SMART GAME 1.0 includes the “Jokerman”, the “Bookman Old Style” and the “Goudy Stout”. Different color mixtures were also used in formatting the text to make them appealing and interesting to the player who happen to be made up of basic level pupils or preparatory school children as the majority.
Below are sample snap shots of texts used in the game:

![Text samples]

4.5 Layers
Now each frame is made up of four major layers. These are the game layer, the background layer, the stop layer and the music layer. The layers are in the form of demarcated strips with circular shapes on them.

4.5.1 Game layer
The game layer is the main layer within the gaming environment. It stores all of the objects and actions that are created by the objects and around the objects. There are blank keyframes within the set of frames in the game layer which provide stability for the game by providing room for
adjustments without tempering with the main structure. All the levels of the game are actually played on the game layer.

Game Layer

4.5.2 Background layer

It is a very important layer in the game programming. This is because it gives a description of the environment in which the game is being played. A game without a background will seem dull and boring. A plain background often works better for simple games. Figure 4.6 shows the background layer panel:

4.5.3 Stop Layer

The stop layer may be regarded as the most essential layer in the game. Without the stop layer the game would be in a chaotic state. This layer runs through all the twenty-four (24) frames making sure that each frame is in a stable playable state.

4.5.4 Sound Layer

The sound layer can be seen as the icing on the cake. This layer helps the player build an interest in the game. Whenever an action is performed it is accompanied by a sound which corresponds to that action. These sounds are funny sounds which are intended to provide humor for the player. The sounds in each frame differ depending on the action, state and stage.

4.6 Pictures and Sounds
These are files made up of a variety of formats and styles. The pictures and sounds depict countless humor characters and backgrounds that are familiar to the player. This is a very useful ingredient in game building. The pictures and sounds vary from frame to frame. The pictures are in “jpg” and “png” formats whiles the sounds are all in “aiff” formats. The pictures and sounds are all imported from different file locations into the library of the flash software. See the figure below samples of picture and sounds used in the game.

4.7 CODES AND INSTRUCTIONS

The coding environment in the Adobe Flash CS5 is similar to that of C++ coding system. The codes vary in function and format. For instance the stop code that is embedded in the stop layer is given by:

stop ();
This code is responsible for keeping all the frames in a stable state so that the game can be played normally.

Another major code which is used in the game is the code that caused the ball to enter from one portal to the other. This code is given by:

```javascript
onClipEvent (enterFrame) {
    if (this.hitTest (_root.char))
    {
        _root.gotoAndStop(4);
    }
}
```

Other codes also perform several functions in the game like the ball character code which is quiet complex.

See the figure below for the ball character code.

Figure 4.9 – Ball Character Code
CHAPTER FIVE
SUMMARY OF GAME OUTLOOK AND RECOMMENDATIONS

5.1 Introduction
This chapter looks at the summary of the game design; recommendations based on the outlook and finally end with the conclusion of the design.

5.2 Summary of Game Outlook
The SMART GAME 1.0 has been tested by several players who think that the game is quiet interesting and is a good achievement with more room for improvement. Since its initial design and testing there has been a series of updates to cater for cheats that arose during the testing of the game. The difficulty level of the game is supposed to rise with the graduation from one level to the other. Few numbers of cheats were identified where the player could bypass the normal mode of playing the game to finish with less effort. These were duly rectified with the introduction of a few adjustments. In a whole the game was a success and a foundation for the creation of more improved versions.

5.3 Recommendations
The development of software like this one has been a very challenging task throughout the project period. Firstly, the time involved to accomplish the task based on the project topic was insufficient. I strongly recommend that the project period for special projects such as this could be given a second look. Also funds need to be provided to support more demanding projects based on the scope.

REFERENCES


**APPENDIX**

**SMARTGAME1.0 CRITICAL CODES**
BALL CHARACTER CODE

onClipEvent (load) { 
  var ground:MovieClip = _root.ground;
  var grav:Number = 0;
  var gravity:Number = 2;
  var speed:Number = 7;
  var maxJump:Number = -12;
  var touchingGround:Boolean = false;
}

onClipEvent (enterFrame) {
  _y += grav;
  grav += gravity;
  while (ground.hitTest(_x, _y, true)) {
    _y -= gravity;
    grav = 0;
  }

  if (ground.hitTest(_x, _y+5, true)) {
    touchingGround = true;
  } else {
    touchingGround = false;
  }

  if (Key.isDown(Key.RIGHT)) {
    _x += speed;
  }

  if (Key.isDown(Key.LEFT)) {
    _x -= speed;
  }

  if (Key.isDown(Key.UP) && touchingGround) {
    grav = maxJump;
  }

  if (ground.hitTest(_x+(_width/2), _y-(height/2), true)) {
    _x -= speed;
  }

  if (ground.hitTest(_x-(_width/2), _y-(height/2), true)) {
    _x += speed;
  }

  if (ground.hitTest(_x, _y-(height), true)) {
    grav = 3;
  }
}
```javascript
RESTART BOX CODE
onClipEvent (enterFrame) {
if (_root.char.hitTest(this)) {
    _root.char._x = charX =
    _root.char._y = charY =
}
}

PORTAL CODE
onClipEvent (enterFrame) {
if (this.hitTest(_root.char)) {
    _root.gotoAndStop();
}
}

BUTTON CODE
on (release) {
gotoAndPlay();
}

ACTION KEY CODE
on (release) {
    gotoAndStop(3);
}

STOP CODE
stop()

LIST OF FIGURES

Figure 4.0 Frame 1 of 24 19
Figure 4.1 Frame Structure 22
LIST OF ABBREVIATIONS

The terms below are used regularly in this project and for the purpose of this study the following shall apply.

APA America Psychological Associations
GES Ghana Education Service
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC</td>
<td>Early Childhood Rating</td>
</tr>
<tr>
<td>ESRB</td>
<td>Entertainment Software Rating Board</td>
</tr>
<tr>
<td>K-A/E</td>
<td>Everyone Rating</td>
</tr>
<tr>
<td>SRA</td>
<td>Software Requirement Analysis</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business Process Modeling Notation</td>
</tr>
<tr>
<td>EEML</td>
<td>Extended Enterprise Modeling Language</td>
</tr>
<tr>
<td>FMC</td>
<td>Fundamental Modeling Concepts</td>
</tr>
<tr>
<td>JSP</td>
<td>Jackson Structured Programming</td>
</tr>
<tr>
<td>SME</td>
<td>Subject Matter Experts</td>
</tr>
<tr>
<td>TDD</td>
<td>Test Driven Development</td>
</tr>
<tr>
<td>UAT</td>
<td>User Acceptance Testing</td>
</tr>
</tbody>
</table>