**CONTENTS**

**CHAPTER 1: INTRODUCTION OF PROJECT………………………………………1-3**

* 1. Abstract , Objective ……………………………………………….……..….1
  2. Introduction…………………………………………………………….…………2
  3. Scope…………………………………………………………………….………..3

**CHAPTER 2: HARDWARE & SOFTWARE REQUIREMENT…………….…………..4**

* 1. Hardware Requirement……………………………………………….…………..4
  2. Software Requirement……………………………………………….…………...4

**CHAPTER 3: TECHNICAL DESCRIPTION………….……………………………..…5-6**

* 1. Front End Description………………………………………………….................5

**CHAPTER 4: SOFTWARE ANALYSIS & DESIGN…………………………….........7-13**

4.1. Software Analysis……………………………………………………………...….7

4.1.1. Software Development Life cycle…………………………………………7

4.1.2. Description of Used Model……………... ………………………………..8

4.2. Software Design…………………………………………………………………………10

4.2.1. Data Flow Diagram………………………………………………………10

4.2.1.1. Zero (0) Level Diagram………….………………………..………. 10

4.2.1.2. One (1) Level Diagram………………………………………………….11

4.2.1.3. Two (2) Level Diagram…………………………………….……….12

4.2.2. System Chart......……….………………………………………………….13

**CHAPTER 5: DEVELOPMENT OF PROJECT……………………………..………14-67**

5.1. Source Code…………………………………………………………………….14

5.2. Snapshots of Output…………………………………………………………….64

**CHAPTER 6: TESTING…………………………………………………………...…...68-70**

**CHAPTER 7: BENEFITS………………………………………………………………….71**

**CHAPTER 8: LIMITATIONS…………………………………………………………….72**

**CHAPTER 9: FUTURE ENHANCEMENT………………………………………...……73**

**CHAPTER 10: CONCLUSION……………………………………………………...……74**

**CHAPTER 11: REFERENCES………………………………………………………...…75**

**CHAPTER 1**

**INTRODUCTION**

**ABSTRACT**

A simplified Shoot the duckgame is developed in this project. The input to the game comes from mouse gestures. The user could move and select the duck to kill it and gain points. The computer program has to handle the gestures of mouse, figure the location of duck, calculate whether the duck can be mouse position and duck position match, calculate the result and present the result to the user as fast as possible.

* 1. **OBJECTIVE**

The ultimate goal of this project is to develop a program implementing “shoot the duck” game . At any time, a user can play this game and entertain himself . The moving ducks at a faster rate moving in bunch prompts the user to select as many ducks and kill them quickly so as to gain more score points. The user friendly and smooth rendering of graphics gives user a rich graphical experience which he embraces.

* 1. **INTRODUCTION**

**Background And Motivation**

Since there are already many high level graphical games around entertaining a complete range of peoples of various age groups. Our project is just a step in recognizing the steps and complexity involved in developing such games.

The problem to deal with is how, the ducks will be moving , where the user points to at particular point of time, whenever the generates the click movement is the duck still in focus or moved ahead, position ducks at different places so as to increase the difficulty of game and maintain interest of the gamer , keep track of the score and do all these all at once without affecting the graphics.

**RELATED WORK**

The first video games were developed in the 1950s, but required mainframe computers and were not available to the general public. Commercial game development began in the 1970s with the advent of first generation video game consoles and home computers. Due to low costs and low capabilities of computers, a lone programmer could develop a full game. However, approaching the 21st century, ever-increasing computer processing power and heightened consumer expectations made it difficult for a single developer to produce a mainstream console or PC game. The average price of producing a video game slowly rose from US$1–4 million in 2000 to over $5 million in 2006, then to over $20 million by 2010. However, mobile, web-based and indie games can cost much less.

Mainstream PC and console games are generally developed in phases. First, in pre-production, pitches, prototypes, and game design documents are written. If the idea is approved and the developer receives funding, a full-scale development begins. This usually involves a 20–100 person team of various responsibilities, such as designers, artists, programmers, testers, etc. The games go through development, alpha, and beta stages until finally being released. Modern games are advertised, marketed, and showcased at trade show demos. Even so, many games do not turn a profit.

**Getting Started.**

**The GUI Framework**

Once computer is setup, GUI framework was decided to work upon which the game will be based on. It’s important to decide what kind of game you are designing before choosing a GUI framework. Each framework offer specific advantages and disadvantages that can severely impact your game. A few of the most common GUI framework options include:

·         jMonkeyEngine – This is a fully functional 3-D engine. If you want to design a 3-D game (such as an RPG), jMonkeyEngine is the way to go. Many of the 3-D features your game needs such as terrain generation are automatically built into this framework.

·         LWJGL – Although not as complex as jMonkeyEngine, LWJGL offers direct access to the OpenGL libraries. The benefit to using this framework is better performance. The drawback is more manual coding to build the game engine from scratch.

·         Swing – Already included in the JRE, Swing is used in games and standalone applications. You should only use Swing for games that are not very graphic intensive. Examples might include a card game or a strategy game. Anything more complex than this is outside the scope of the Swing framework.

**Deploying the Game**

Another thing that’s great about Java games is that you have many choices when it comes to deployment. Unlike games written in C, a Java game can be downloaded and played locally on a user’s computer, played directly in a web browser (Applet), or a combination of the two which is known as a Java Web Start. A Web Start downloads most of the resources locally but is initiated from a browser window.

You could also decide to make your game completely Web based using HTML5. In this case, the entire game is run on the server side and can be useful for multiplayer web games.

No matter which deployment method (or combination) you choose, the key to developing an awesome Java game is allowing as many people as possible to play it.

Java is a very popular platform and allows game to reach as many potential customers as possible. In addition to being a fun and interesting learning experience, creating Java games can become a full-time occupation with a little bit of effort and some creativity.

**SCOPE**

User does not need an internet connection to access this software the only requirement to use this software is desktop with very low configuration. all users will be provided with user manuals for the new hardware they receive and any applications that required remediation and functionality was either changed or added. There will also be training provided for the new operating system. Peripheral equipment purchases no other hardware devices will be purchased as a part of this project. Provide Access to all the major configuration tools. Also provides access to all the important settings. Software has very user friendly interface which is very easy to handle and understand . Since there are already many high level graphical games around entertaining a complete range of peoples of various age groups. Our project is just a step in recognizing the steps and complexity involved in developing such games.

The problem to deal with is how, the ducks will be moving , where the user points to at particular point of time, whenever the generates the click movement is the duck still in focus or moved ahead, position ducks at different places so as to increase the difficulty of game and maintain interest of the gamer , keep track of the score and do all these all at once without affecting the graphics.

**HARDWARE & SOFTWARE REQUIREMENT**

**SYSTEM REQUIREMENT**

**2.1 HARDWARE REQUIREMENT**

* Pentium IV processor or higher
* 128 MB RAM(or above)
* 40 Gb HARDDISK
* Mouse/Keyboard

**2.2 SOFTWARE REQUIREMENT**

* OS-Windows XP
* Jre Environment 1.5 or higher
* Jdk , CSS
* Notepad/Netbeans

**TECHNICAL DESCRIPTION**

**3.1 FRONT END DESCRIPTION**

**LANGUAGE:- JAVA**

Java is a programming language originally developed at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte code that can run on any Java virtual machine (JVM) regardless of computer architecture

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any supported hardware / operating-system platform. One should be able to write a program once, compile it once, and run it anywhere. This is achieved by compiling the Java language code to an intermediate representation called Java byte code, instead of directly to platform-specific machine code. Java byte code instructions are analogous to machine code, but are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets.

Java uses an automatic garbage collector to manage memory in the object lifecycle. The programmer determines when objects are created, and the Java runtime is responsible for recovering the memory once objects are no longer in use. Once no references to an object remain, the unreachable memory becomes eligible to be freed automatically by the garbage collector. Something similar to a memory leak may still occur if a programmer's code holds a reference to an object that is no longer needed, typically when objects that are no longer needed are stored in containers that are still in use. If methods for a nonexistent object are called, a "null pointer exception" is thrown.

One of the ideas behind Java's automatic memory management model is that programmers be spared the burden of having to perform manual memory management. In some languages memory for the creation of objects is implicitly allocated on the stack, or explicitly allocated and de-allocated from the heap. Either way, the responsibility of managing memory resides with the programmer. If the program does not de-allocate an object, a memory leak occurs. If the program attempts to access or de-allocate memory that has already been de-allocated, the result is undefined and difficult to predict, and the program is likely to become unstable and/or crash. This can be partially remedied by the use of smart pointers, but these add overhead and complexity. Note that garbage collection does not prevent 'logical' memory leaks, i.e. those where the memory is still referenced but never used.

Garbage collection may happen at any time. Ideally, it will occur when a program is idle. It is guaranteed to be triggered if there is insufficient free memory on the heap to allocate a new object; this can cause a program to stall momentarily. Explicit memory management is not possible in Java.

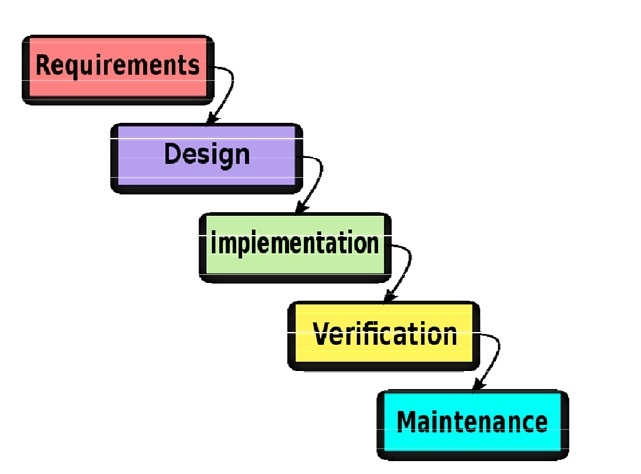
Java does not support C/C++ style pointer arithmetic, where object addresses and unsigned integers (usually long integers) can be used interchangeably. This allows the garbage collector to relocate referenced objects, and ensures type safety and security.

As in C++ and some other object-oriented languages, variables of Java's primitive data types are not objects. Values of primitive types are either stored directly in fields (for objects) or on the stack (for methods) rather than on the heap, as commonly true for objects (but see Escape analysis). This was a conscious decision by Java's designers for performance reasons. Because of this, Java was not considered to be a pure object-oriented programming language. However, as of Java 5.0, auto boxing enables programmers to proceed as if primitive types were instances of their wrapper class.

**SOFTWARE ANALYSIS&DESIGN**

**4.1 Software Analysis**

**4.1.1 Software Development Life cycle**

****

**Fig:- SDLC**

**4.1.2 DESCRIPTION OF USED MODEL**

**The Waterfall Model**

The Waterfall model is a sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design (validation), Construction, Testing and Maintenance.

To follow the waterfall model, one proceeds from one phase to the next in a sequential manner. For example, one first completes requirements specification, which after sign-off are considered "set in stone." When the requirements are fully completed, one proceeds to design. The software in question is designed and a blueprint is drawn for implementers (coders) to follow — this design should be a plan for implementing the requirements given. When the design is fully completed, an implementation of that design is made by coders. Towards the later stages of this implementation phase, separate software components produced are combined to introduce new functionality and reduced risk through the removal of errors.

Thus the waterfall model maintains that one should move to a phase only when its preceding phase is completed and perfected. However, there are various modified waterfall models (including Royce's final model) that may include slight or major variations upon this process. Time spent early in the software production cycle can lead to greater economy at later stages. It has been shown that a bug found in the early stages (such as requirements specification or design) is cheaper in terms of money, effort and time, to fix than the same bug found later on in the process. To take an extreme example, if a program design turns out to be impossible to implement, it is easier to fix the design at the design stage than to realize months later, when program components are being integrated, that all the work done so far has to be scrapped because of a broken design.

This is the central idea behind the waterfall model - time spent early on making sure that requirements and design are absolutely correct will save you much time and effort later. Thus, the thinking of those who follow the waterfall process goes, one should make sure that each phase is 100% complete and absolutely correct before proceeding to the next phase of program creation. Program requirements should be set in stone before design is started (otherwise work put into a design based on incorrect requirements is wasted); the program's design should be perfect before people begin work on implementing the design (otherwise they are implementing the wrong design and their work is wasted), etc.

A further argument for the waterfall model is that it places emphasis on documentation (such as requirements documents and design documents) as well as source code. In less designed and documented methodologies, should team members leave, much knowledge is lost and may be difficult for a project to recover from. Should a fully working design document be present (as is the intent of Big Design Up Front and the waterfall model) new team members or even entirely new teams should be able to familiarize themselves by reading the documents.

Basic principles of the waterfall model are:

Project is divided into sequential phases, with some overlap and splash back acceptable between phases.

Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.

Tight control is maintained over the life of the project through the use of extensive written documentation, as well as through formal reviews and approval/signoff by the user and information technology management occurring at the end of most phases before beginning the next phase.

**4.2 SOFTWARE DESIGN**

**4.2.1 Data Flow Diagram**

The DFD takes an input-process-output view of a system. That is, data objects flow into the software, are transformed by the processing elements, and resultant data objects flow out of the software. Data objects are represented by labeled arrows and transformations are represented by circles. The DFD is represented in a hierarchical fashion. The first DFD represents the system as a whole. Subsequent data flow diagrams provide increasing detail with each subsequent level.

The data flow diagram enables the software engineer to develop models of the information domain and functional domain at the same time. As the DFD is refined into levels of greater detail, the analysts perform an implicit functional decomposition of the system. Also DFD refinement results in a corresponding   
refinement of data as it moves through the processes that embody the application.

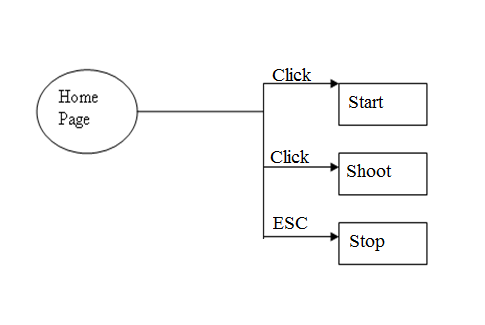
**Play**

**USER**

**OUTPUT**

**Fig:- Zero (0) Level**

**4.2.1.2 DATA FLOW DIAGRAM**

****

**Fig:- One (1) Level**

**DEVELOPMENT**

**5.1 SOURCE CODE**

**CONCEPT**

The concept behind the working of this software is to use the control commands of the system and then run them through java. Well as already discussed in analysis section about control commands that they are system defined commands and can be use to access every setting of the system. So we used java to run these commands. Java provides to make a runtime environment and to run exe files or system commands to build a process. Also it provides a way to handle the process and to do operations on them.

So we first created a environment to handle the process and then we create the process of the command selected by the user. So in foreground user just clicking the option but in background the respective control command is firing.

**SOURCE CODE**

**5.2 SNAPSHOTS**

**FRONT PAGE**

****

****

****

**CHAPTER 6**

**TESTING**

**6.1 TESTING**

Testing is more than just debugging. The purpose of testing can be quality assurance, verification and validation, or reliability estimation. Correctness testing and reliability testing are two major areas of testing. Software testing is a trade-off between budget, time and quality.

**Software Testing**

Software Testing is the process of executing a program or system with the intent of finding errors. Or, it involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Software is not unlike other physical processes where inputs are received and outputs are produced. Where software differs is in the manner in which it fails. Unlike most physical systems, most of the defects in software are design errors, not manufacturing defects.

**To improve quality**

As computers and software are used in critical applications, the outcome of a bug can be severe. Bugs can cause huge losses.

**For Verification & Validation (V&V)**

Another important purpose of testing is verification and validation (V&V). It is heavily used as a tool in the V&V process. Testers can make claims based on interpretations of the testing results, which either the product works under certain situations, or it does not work.

**Software Testing Types**

**Black-box testing**

The black-box approach is a testing method in which test data are derived from the specified functional requirements without regard to the final program structure. It is also termed data-driven, input/output driven or requirements-based testing. A testing method emphasized on executing the functions and examination of their input and output data.

**White-box testing**

Contrary to black-box testing, software is viewed as a white-box, or glass-box in white-box testing, as the structure and flow of the software under test are visible to the tester. This testing is based on knowledge of the internal logic of an application’s code. Testing plans are made according to the details of the software implementation, such as programming language, logic, and styles. Test cases are derived from the program structure. White-box testing is also called glass-box testing, logic-driven testing or design-based testing .

**Unit testing**

This involves testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code.

**System testing**

Entire system is tested as per the requirements. Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system.

**End-to-end testing**

Similar to system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

**Usability testing**

User-friendliness check. Application flow is tested, Can new user understand the application easily, Proper help documented whenever user stuck at any point. Basically system navigation is checked in this testing.

**Install/uninstall testing**

Tested for full, partial, or upgrade install/uninstall processes on different operating systems under different hardware, software environment.

**Recovery testing**

Testing how well a system recovers from crashes, hardware failures, or other catastrophic problems.

**Security testing**

Can system be penetrated by any hacking way. Testing how well the system protects against unauthorized internal or external access. Checked if system, database is safe from external attacks.

**Compatibility testing**

Testing how well software performs in a particular hardware/software/operating system/network environment and different combination s of above.

**Comparison testing**

Comparison of product strengths and weaknesses with previous versions or other similar products.

**Alpha testing**

In house virtual user environment can be created for this type of testing. Testing is done at the end of development. Still minor design changes may be made as a result of such testing.

**Beta testing**

Testing typically done by end-users or others. Final testing before releasing application for commercial purpose.

**CHAPTER 7**

**BENEFITS**

**7.1 BENEFITS**

* Easy to operate.
* One Click Start and Stop.
* Software has very user friendly interface which is very easy to handle and understand.
* Platform independent.
* Software uses very less memory and takes less time to startup.

**CHAPTER 8**

**LIMITATIONS**

**8.1 LIMITATIONS**

* Only One Game Stage i.e. single screen.
* System requires jre installed on the system.
* Game Pause Option in not available in current version.
* GUI is in English only.

**FUTURE ENHANCEMENT**

**9.1 FUTURE ENHANCEMENT**

* More Game Stages will be provided.
* Pause option can be included in this software..
* This software can be developed to use as tutorial to teach basic concepts of Game software to new users.
* Multi user facility is included in this software.

**CHAPTER 3**

**CONCLUSION**

**10.1 CONCLUSION**

This project’s goal is to “Shoot the duck” a real time graphical UI game. The report gives details in the game rendering methods for the implementation of the game. We have tested the program on several different users and the results were satisfying. The proposed algorithm is based on several screen sizes, and different processing environments. This project focused mainly on mouse motion recognition and dealt with position of duck- figuring when is the duck killed and maintain the score. The method at hand can be broaden and applied not only for the above specific game, but also for other graphical games, depending on other experiences requires to be built upon. Moreover, an obvious advantage using Canvas is that it provides rich and glib experience. By defining appropriate speed, we provide fast and swift movements of duck and mouse so as to maintain user interest.

The main drawback of our program is the fact that it depends on mouse movements and user clicks so the experience is not the best on laptops.

**REFERENCES**

**11.1 REFERENCES**

* Wikipedia
* www.java.sun.com/docs/books/tutorial
* Win Xp Explore book
* Wrox professional Java
* Kathy Sierra. Head first Java
* www.StudyStuff.in