Improved Iterative Software Development Method for Game Design

by

Umair Azfar Khan

University of Tampere
Department of Computer Sciences
Computer Science
M.Sc. Thesis
Supervisor:
Martti Juhola
April 2008

University of Tampere
Department of Computer Sciences
Computer Science
Umair Azfar Khan
Improved Iterative Software Development Method for Game Design
M.Sc. thesis, 53 pages
April 2008

This paper describes a game making and editing tool which simplifies the process of game making by removing the need to recompile the code. For this reason a simple side scrolling game engine was developed which also supports an inbuilt editor. With the help of this editor all the objects present in the game engine can be changed to look the way a user wants them to. This reduces the communication gap that is normally found in between the coding and art development team for a game. The artists are now given the liberty to draw the objects by themselves and that is automatically integrated into the game.

In order to save all the changes done to the objects, XML files are used, which store all the information necessary for drawing an object and its constituent sub-objects. These XML files can be worked on separately by the users and then later exchanged which speeds up the whole development process and also provides the functionality to share each other's work. Thus, the game engine remains the same, but the same engine can be used to create two totally different games that look totally different from each other.

Key words and terms: Game, editor, object, recompile.

Dedications

This thesis is dedicated to my mother who passed away during the time I was completing my studies. She always believed in me through all the decisions that I made and supported me whenever I was overburdened with work and low on spirits.

This is also dedicated to my father who always kept my studies at a higher priority than anything in my life. He always kept me focused on what was important and what took preference over everything else.

Finally, I would like to dedicate this thesis to all the friends and family who understood all the problems that I had been facing and provided support and comfort through all the hard times.

Acknowledgements

Dr. Martti Juhola has been the ideal thesis supervisor during the time I had the privilege in working under him. His insightful criticisms, encouragement and all out support have helped me in innumerable ways in completing this thesis. I really appreciate all his effort in bringing this thesis to its completion.

Contents

1. Introduction

The software industry has been lately taking a lot of interest in the end-user interaction and software development. The aim has been to bring the end-user closer to the whole software development process, without complicating the whole development process. The bottleneck in this effort is the constant requirement of the software to be recompiled whenever a change is made. This is the reason why once software is released the development cycle does not end there. A series of patches and fixes are released constantly to make changes and fix many bugs within the software.

These days, great care is being taken to make the software development process as humanly intuitive as possible. The problem here is that, the configuration and compilation of the code requires the user to have some programming experience and the installation of different development environments to get the job done. This requirement already forces the users of the application out of the end-user zone to casual programmers.

The aim of this **Ga**me **M**aking and **Edi**ting **T**ool (Gamedit) is to narrow the gap between software development and the end-user. Gamedit provides a simple 2D side scrolling game with an in-game editor that lets the user define his own objects being used by the game engine. These objects include all the menus and the objects that the user can interact with while playing the game.

The editing is done by using an in-game editor that behaves like a simple drawing tool that people are normally used to. This provides an intuitive way of making changes to the software and no code needs to be recompiled for this to work properly. Whatever you draw on screen is what you get inside the game, thus minimizing the time required in between editing the game objects and reproducing the change inside the game.

Gamedit improves upon the iterative software development method for 2D games by allowing the testers to play the part of the coder. The testers are provided with an in game editor which avoids the software to go under unnecessary iterations.

2. Background

Computer games have been a part of the mainstream software ever since the days of early personal computers. Programming games has always been considered one of the toughest jobs to do in the field of computer science. Every game comes with its complex problems which normally require a complete overhaul of the way the game is to be made or the problem is to be solved. This is the reason why, most of the times, the programmers are so busy in perfecting the solution to the various problems that they forget how to make the game more fun and to reach out to broader audience than the hard core gamers.

This trend has somewhat changed lately when almost every game that is released, comes with its own software developers kit and development tools which allow the gaming community to experiment with the game engine and make changes as they see fit. The problem with this approach is that only those users who have some programming experience can use the tools. This leaves out a great deal of users who want to make changes to the software, but do not have the expertise to do so.

Another problem that comes up is the constant dependency of the software to be recompiled with every little change that is made to the code. The code once compiled does not provide any dynamic interaction that might change it from within. Code needs to be recompiled to see if the change that was made has produced the required result or not. This is another time consuming process that increases the time in development.

What is required is software that can be changed at runtime. This means that once a program has been created, it can be changed from within from inbuilt tools. Thus, the program is intelligent enough to read the changes without the need to recompile the entire code.

With the arrival of object oriented programming languages such as JAVA, C#, C++, etc., making a software that changes itself has become both easier to code for and to manage. Object oriented programming languages help define the entire software in a series of objects where each object can be self contained and the change done on one object can be reflected on to many similar objects. This helps in separating the code blocks from one another and whatever makes programming easier can be reflected on how the program behaves when it is edited from within. In order to bring the editing tool to the broader audience, it is supposed to be:

- Intuitive enough to let the users learn it from the get go
- Accessible enough so that the users can use it any time of the day and,
- Simple enough to hide the unnecessary details that make the software daunting

The aim for Gamedit was to complete all these requirements and provide a framework for all the future software to base their development on. It should be noted that Gamedit is not merely a game, but an editing tool that changes the software without recompiling. A game engine is used because it provides the quickest way to see the changes in the software as the user is dependent on the visual responses to enjoy the game.

2.1. Selecting the right hardware:

One of the main concerns about making software accessible to the general users is to have it in a device that is in every day use. Life nowadays demands users to be on their feet and mobile, which makes their time spent on a computer lesser day by day. This is the reason why the mobile devices have become so popular recently, hence increasing the demand in functionality of these devices.

Keeping this in mind, the target hardware for Gamedit was mobile phones and in the mobile phones, that category which supports Java for mobiles. Java Platform, Micro Edition or J2ME [Java ME] provides a flexible environment for running applications on mobile phones and other embedded devices. This has the advantage of running Gamedit on the maximum number of mobile devices as Java is platform independent; hence, a maximum number of users can get a hold of the software on devices that they use daily. This satisfies the condition where the users can use the software on a daily basis.

Another important thing about mobile devices is the limited number of keys available to the user to use any mobile application. This makes it necessary that each application is designed to use the least number of keys and be intuitive enough so that every user can start using it almost instantly. This in itself provides the restriction that the application must be easy enough for every user to use.

2.2. Selecting the IDE and Language

For creating Gamedit, I chose to use the Eclipse IDE [Eclipse]. It has an EclipseME plugin [Eclipse ME] that supports J2ME development using the Eclipse environment. I have a working experience in J2ME from my school years and then the job that I was doing for the past 1 year. So picking Eclipse and J2ME as the development environment came naturally. In order to use the IDE, the Sun Java Wireless Toolkit for CLDC [Sun Java Wireless Toolkit] needs to be installed. The version used was Sun Java Wireless Toolkit 2.5.2 for CLDC along with J2ME Polish [J2ME Polish], which is a suite for creating enhanced J2ME applications.

J2ME Polish contains PDA optional packages (JSR 75) which gives developers the FileConnection API used for accessing the mobile phone's file system. Using this optional package was critical in Gamedit's development as file reading and writing plays a major role in saving the user's progress and loading the old settings done by a user.

Another reason for using J2ME was the ease of development of mobile applications. I wanted the application to support easy game development which J2ME handles really well. Making a game was a priority as a game provides the most visually accessible feedback for user responses. The ease of programming provided the option of getting the most out of the least amount of code, which helped in creating the Gamedit application faster so that the whole idea can be tried and tested easily and quickly.

2.3. Idea and Inspirations

The idea of this thesis came up while working on my job one day. I visualised a game where the puzzles can be solved by drawing the objects in a game. This called for a proper physics engine to be implemented, plus a dynamic game environment, where objects can be added and removed by drawing them. Later the idea was further enhanced by envisioning a tool which lets the user draw the art of the game rather than pre-built programmer defined objects. This called for developing a game which had the least amount of rules hard coded, while the rest of the game defined by the users themselves. This created the requirement for making a game engine which needs to be compiled just once and the rest can be done from within the game. Hence the idea transformed into what is now known as Gamedit.

While developing Gamedit, I came across many similar ideas that proved to be both fascinating and inspirational. Their emphasis was more on using user's feedback with the in-game physics to achieve a result, while my idea had transformed into redefining the whole game from within the game itself. The first inspiration was Phun, which is a Master of Science Thesis by Computing Science student Emil Ernerfeldt at VRLab, Umeå University, Sweden [Ernerfeldt, 2008]. The other inspiration was Crayon Physics Deluxe, which was the finalist in the Independent games Festival of 2008 [Crayon Physics Deluxe]. Surprisingly, both these ideas come at a time when I myself was working on a similar project, but I do think that my idea stands separate from what these other topics have tried to achieve.

3. Traditional Software Development

The traditional software development processes have always contained a series of weaknesses and problems which has asked for continuous evolution and development of old and new development models time and again. The sources for most of the development processes lie in the theoretical concepts applied as well as the use of inadequate cost analysis models. One of the major reasons for the inadequacy of the traditional processes is the improper participation of the several groups that are related to the development of the software, one way or the other. Rauterberg argues that there are three essential barriers to optimizing the software development process [Rauterberg, 1992]: the specification barrier, the communication barrier and the optimization barrier.

3.1. Specification Barrier

The requirements specification has always been a major problem whenever it comes to defining software. The main reason for this is normally the gap between what the client understands and what is actually doable under the budget that the software developers need to work in. Another reason is for the client to understand what part of the project can be done based on the technology in question. A software developer can not create real-time animation using just HTML for example.

There is also no way to ascertain that once the requirements have been written down and the development team starts designing and implementing the software, those requirements will not changed. This is the reason why many early software development models such as the Waterfall model are generally not used these days. A more open approach such as the incremental or iterative software development is used. Even Agile software development methods are used as they provides the coders with a flexible set of requirements and much is decided on the feedback that is received from the clients after some on hands time with the developed prototype. It is therefore necessary to find other informal, semi-formal to formal methods for requirement specification.

It can also be catastrophic to assume that the clients, usually the people from middle or higher levels of management, are able to provide adequate information for any software system. Hence, the following perspectives should be taken into consideration during the software analysis and specification phases [Rauterberg, 1992]:

3.1.1. The Applier's Perspective

Every person who can help in defining the requirements for software system is considered to be an applier. Normally, the clients hold the role of the applier when it comes to defining the perspectives. This perspective takes into consideration all the general requirements concerning the organizational structures, project costs, and implementation goals for the complete software system.

3.1.2. The User's Perspective:

Users are those persons who need the software to produce results for performing their tasks. The major factor that influences their perspective is human-to-human communication with the end-users such as their bosses or department head. Their contribution normally is to define requirements for the software interface.

3.1.3. The End-User’s Perspective:

End-users are all those people who use the software system directly as a work tool. This group formulates the essential requirements for the tool and the input/output interfaces.

3.2. Communication Barrier

This barrier has different layers that need to be overcome to minimise the unnecessary requirements of the software system. The first layer is the difference of opinion, information and understanding between the three groups of people described in the above mentioned perspectives. The way the requirement trickles down from the appliers, to users and finally to the end-user and then from end-user to applier, many things may go wrong or maybe misunderstood.

The second layer comes between the difference in understanding between the client and the software development team. The understanding of the software developers with their technical jargon is of no meaning to the client. Many times a client may want a simple solution to a problem, whereas the developers might make it too complicated than it really is. Sometimes the client might not be able to relax the requirement, even a little bit, that might make the software very difficult to use. I personally had a similar experience while working during my days at Jintech Pvt Limited, Pakistan, where the client wanted the website for their product to work exactly as their software used to work. This made the code for their website extremely complex and hard to do.

3.3. Optimisation Barrier

The final barrier is the optimisation barrier, which ascertains as to which part of the project requires more optimisation and how that optimisation is needed to be carried out. Software might be broken into several parts and each part is needed to be optimised separately or maybe prototypes are needed to check the user response to the different parts of the software and each part is to be optimised separately. Overcoming this barrier can help in reducing the development cost and help in targetted software development.

3.4. Methods Overcoming the Barriers

Time and again, there have been many software development methods that have been introduced to overcome the barriers as mentioned above. These methods have had different degrees of success when it came to developing software with changing requirements. Most of these methods require close communication between the client and the software development team which minimises the understanding gap which is normally encountered during other software development methods. A brief mention of some of these techiniques is as following:

3.4.1. Prototyping

A prototype is used to acquaint the end-users with the procedural character of the system being developed. Prototyping is used to display the working of a part of or the entire application system to the end-user so that he is able to grasp the way the entire system is going to work. Thus prototyping provides a particularly effective means of communication between the user and the developer as a small trailer of the entire application is presented to the user and the user feedback is acquired readily and incorporated into the future releases of the prototype.It is however extremely important that the time taken from acquirring user feedback on the prototype to the implementation of the new prototype is short.

3.4.2. Agile Software Development

The idea behind Agile software development is to complete the entire software development cycle in one small iteration. The iteration may not necessarily contain all the workings of the final software but it should have some working software which was intended at the end of that iteration. Normally this release of software, also known as an iteration, lasts 4 weeks. Software developed during one unit of time is referred to as an iteration, which may last from one to four weeks. Each iteration is an entire software project, which includes planning, requirements analysis, design, coding, testing, and documentation.

3.4.3. Iterative and Incremental Software Development

Incremental software development requires that different parts of the entire software system are developed at different times or rates, and integrated to the whole as they are completed. During this type of software development some time is set aside to revise and improve parts of the completed system. In Iterative software development, the user feedback that is received after each iteration is not used as input for revising the plans or specifications of the successive increments. This however might be used for modifying, and especially for revising the targets of each successive iteration.

4. History of Iterative Software Development

Iterative software development is the method under discussion in this thesis. It is only fair that we begin from the history of this development method and move onto its usability in the game design process. In order to grasp the idea of iterative software development, we need to start from the basic software development models and move up to their evolution over the years which, in turn, have provided us with the iterative development model. But before beginning, it is necessary to clarify the difference between an increment and iteration. [Goldberg, et al., 1995] has defined this difference as:

- Iteration: The controlled reworking of part of a system to remove mistakes or make improvements.
- Increment: Making progress in small steps to get early tangible results.

4.1. Basic Software Development Models

The main reason for using a software development model is to produce better quality software with complete documentation quickly and efficiently. The goal is the same for all the software models that are in use today but the end result is not always the same. Barry Boehm [Boehm, 1988] states that the primary function of any software model is to determine the stages in software development and the order in which they are going to be implemented for developing complete software; this also includes the information about the transition criteria that is needed to progress from one stage to the next. The completion criteria and the entrance criteria for the next stage are a part of these transition criteria.

Many software projects in the past have experienced serious problems because there was no systematic way of completing the various development activities. No proper systematic way or model was used, which caused exceptional loss in man hours and effort once the initial design needed to be reworked in order to make the software work properly. As a result, a number of software development models have been employed throughout the industry with different degrees of success. All these models can be grouped under two main categories; the waterfall model and the evolutionary model.

4.1.1. Waterfall Models

The most popular software development model is the waterfall model. In this method, the feedback loops are confined to successive stages instead of reworking all the stages once one iteration of software is complete. This means that once a stage has been complete, there is no turning back and the software moves to the next stage of development.

The assumptions on which the waterfall model is based on are:

- The customer knows what is required off the end product
- Once the requirements have been finalised in the requirements specification phase, they cannot be changed
- Phase reviews are used as control and feedback points

The characteristics of a successful project completed using the waterfall model are:

- The requirements have remained stable during the completion of the project
- The environments have remained stable
- Focus has remained on completing the project as a whole
- In the end there is one final product delivery

The stages that are present in the waterfall model can best be described by Figure 1 as given by [Boehm, 1988]:

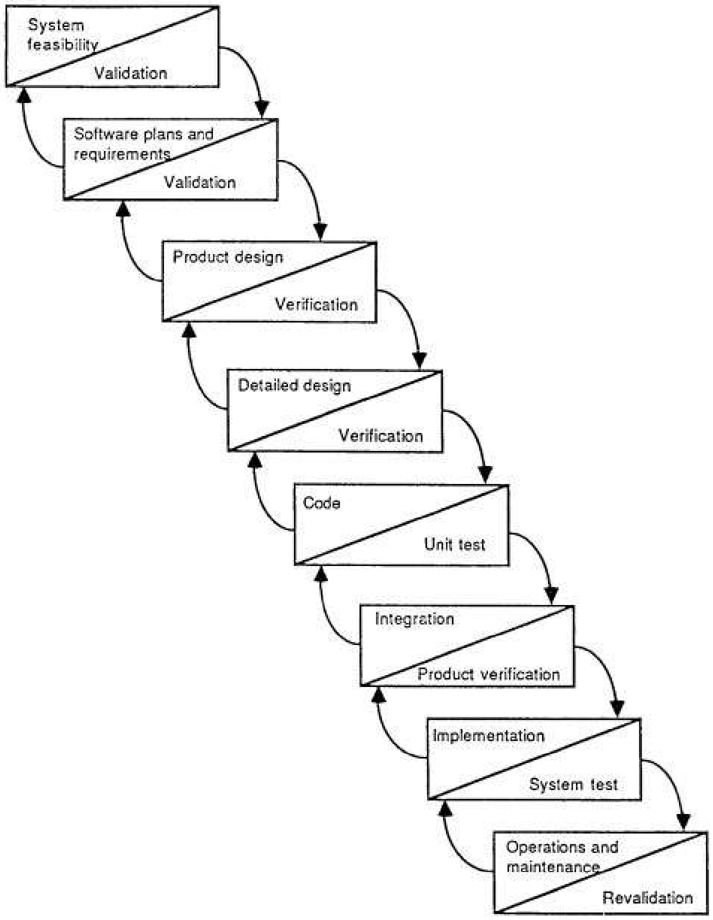

Figure 1 – Stages of a Waterfall Model [Boehm, 1988]

The goal for each stage of a waterfall model is to produce a fully elaborated document in the end. This works as the completion criterion for the current stage and once this document is accepted, as the entry criterion for the next stage. Even though this form of trickle down approach for completing different phases of the project and then moving on to the next has been successful for some projects, it has not been successful for those projects that required extensive end user interaction. Hasan Savani [Savani, 1988] has best described it as:

"The waterfall model works well for salmon, not people. While trying to achieve their goal, salmons tend to move intractably upstream. When people strive for a goal they tend to meander, sometimes venturing forward and sometimes retracting steps. While following a general course from the problem to the solution, people often look ahead (such as by prototyping or simulation) and then retreat to accommodate their current understanding of the problem to what they learned in their look ahead."

4.1.2. Evolutionary Models

The evolutionary model is characterised by frequently occuring feedback loops and extensive customer participation in every iteration of the software as it is completed. In this model, with the completion of each increment all the stages are expanded to incorporate changes as defined by the customer after testing. Thus, with each increment the software gets better and the customer is kept satisfied.

The assumptions for the evolutionary model are:

- The customer cannot be sure of what he wants, hence close communication is a necessity.
- The requirements will change with time.
- Feedback and control should be acquired through continuous reviews.

The characteristics of a successful evolutionary model project are:

- Acquisition of continuous customer feedback
- The targets for the product are not entirely fixed
- Focus is always on the most important features
- The software has frequent releases

The evolutionary model as given by [Martin, 1991] is best described by Figure 2.

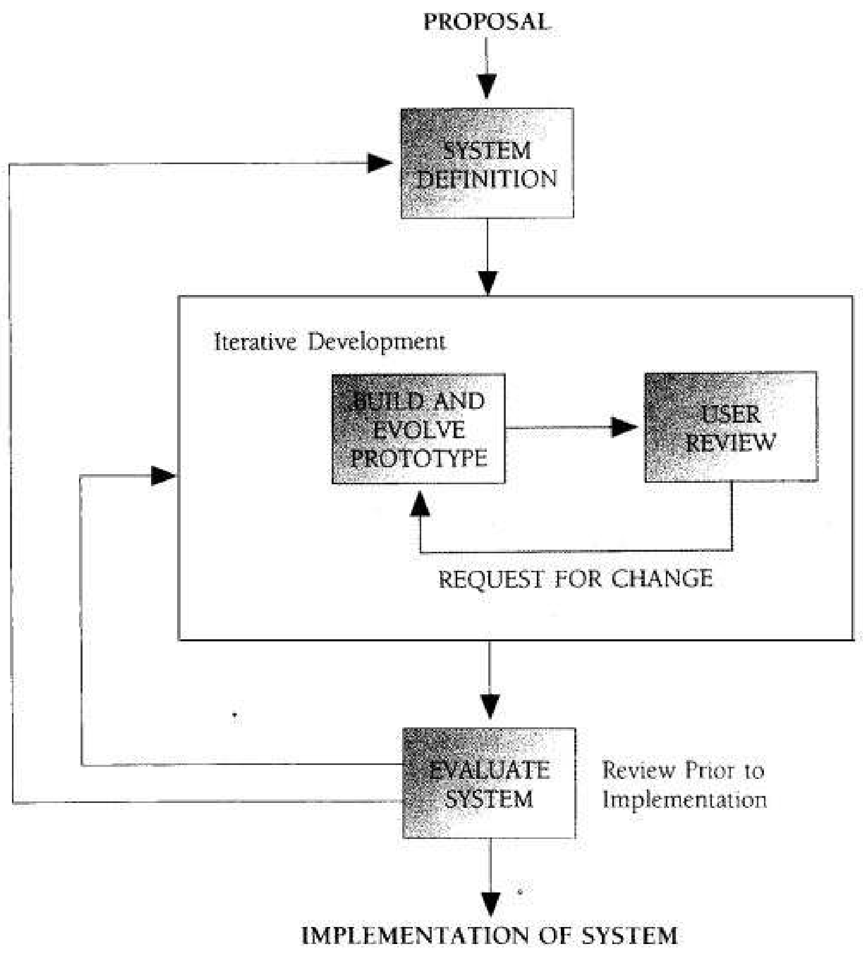

Figure 2 – Stages in an Evolutionary Model [Martin, 1991]

Another name for evolutionary models is the Rapid Application Development Models or RAD models for short. These models are perfectly suited for devloping applications which require close communication with the end-user. The requirments can only be established after the end-user has had some hands on time with the application, which generally is a prototype in this scenario.

This doesn't mean that the evolutionary models do not have a problems of their own. If a software is being developed for a wide market which has a large variety of customers, the software might go through endless increments and making a final product may become problematic. Also, since the product is constantly changing, it might make the software unstable and might require a special increment just to stabilise it. Before we move to define the iterative software development we should explain the various evolutionary models that are being used for development. These models are the spiral model, the RAD model and the agile models.

Spiral Model:

The Spiral Model was defined by Barry Boehm [Boehm, 1988] which aims to incorporate the details of both the waterfall and the evolutionary model. In the spiral model, full cycles of development are used. These successively refine the end product. The order of these cycles is risk driven. Simulations and prototypes are used in the early cycles to evaluate all the alternatives and to resolve risks. These are then concluded with reviews and approvals of completed documentation before the next cycle is started. Once all the risks have been covered, the last cycle is initiated, which is the conventional waterfall development of the product.

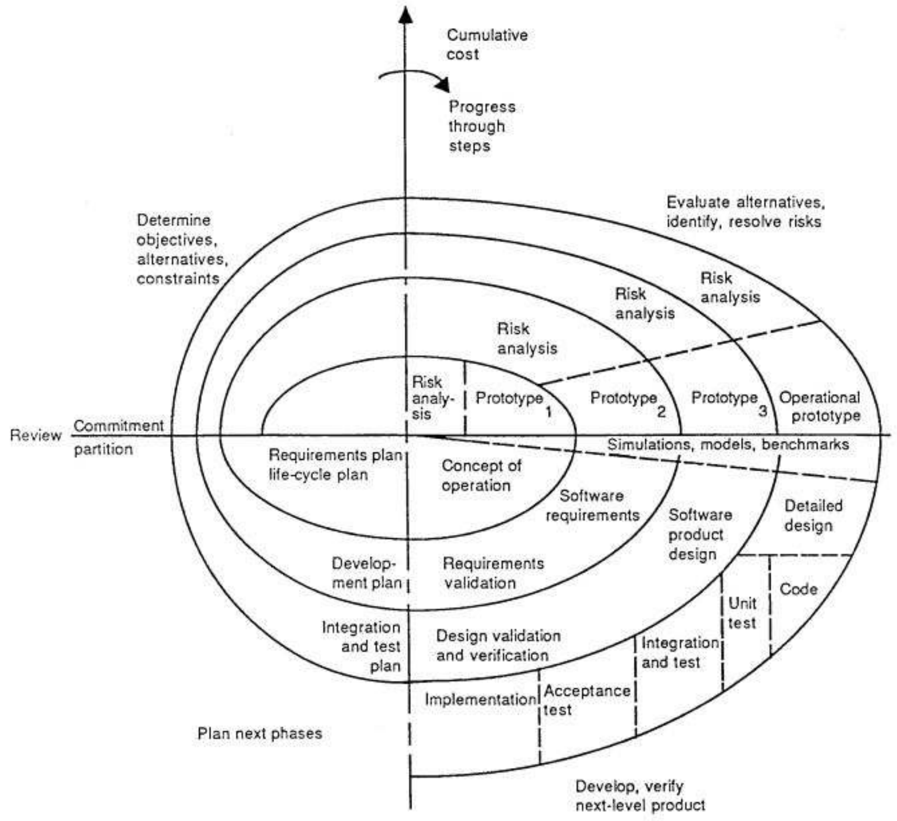

Figure 3 - The Spiral Model [Boehm, 1988]

4.1.3. The RAD Model

The original Rapid Application Development (RAD) model as introduced by James Martin [Martin, 1991] was based on short incremental cycles based on business priorities and required close involvement of the customers. Each increment of the RAD model is a small waterfall model developed by a SWAT (Skilled Workers with Advanced Tools) team. The SWAT Team is a small team of two to six developers who are specialized in working together with the development tools and produce builds of the software at high speed. Their goal is to create fast deliverables after each iteration by taking the user feedback into consideration. It is with the effort of this team that the users get an early build to test and provide comments which are extremely important in creating the final application. Since the conception of this idea, it has become a generic term for many different types of evolutionary software development models.

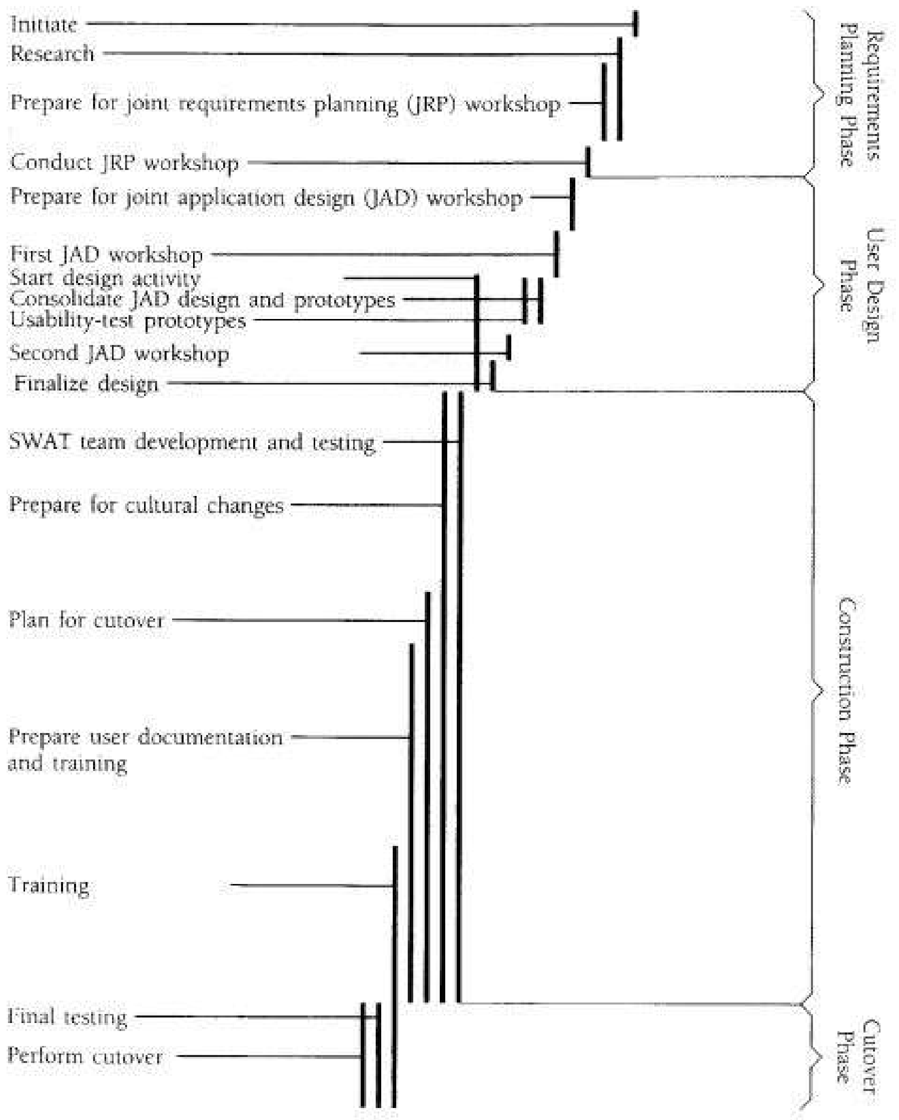
Initiate
Research
Prepare for joint requirements planning (JRP) workshop
Conduct JRP workshop
Prepare for joint application design (JAD) workshop
First JAD workshop
Start design activity
Consolidate JAD design and prototypes
Usability-test prototypes
Second JAD workshop
Finalize design
SWAT team development and testing
Prepare for cultural changes
Plan for cutover
Prepare user documentation and training
Training
Final testing
Perform cutover
Requirements Planning Phase
User Design Phase
Construction Phase
Cutover Phase

Figure 4 – Rapid Application Development Model [Martin, 1991]

As seen in Figure 4, RAD depends on joint activities like Joint Requirements Planning (JRP) or Joint Application Design (JAD) to complete the software. Joint Requirements Planning (JRP) is a technique where the software professionals hold joint planning sessions for creating user requirements. These are informal sessions or workshops aim to provide an open environment for people to discuss their responsibilities and the critical information that they require for carrying out their job.

At the end of each JRP session, a Written documentation defining the requirements is produced. There are many benefits that are provided by the JRP workshops such as,

- Encouraging the development of a partnership between business and software experts.
- Identification of the software needs that the business side will be able to fulfil with the software.
- Clarifying all software requirements which reduce the overall design and development time.
- Driving the decisions pertaining to software architecture and platform.
- Issues are resolved early in the system life cycle which lowers the deployment and maintenance costs.
- Combines the ideas of a variety of people which improves the quality of the solution
- Increasing the knowledge of the end user and project team about the system.

JRP and JAD are often used synonymously due to the identical nature of both the processes. The only difference is that where JRP is more centred towards acquiring requirements with stakeholder's feedback, JAD is used for redesigning the application. Due to the non-constant nature of the Rapid Application Development process a change in requirement normally means a change in design, hence both terms can be used interchangeably.

4.1.4. Agile Software Development Models

Agile software development can also be explained as adaptive software development. The idea is to create software in a matter of weeks rather than months, consult with the user at every step and adapt to the changes as requested by the client. It should be noted that agile software development is considered to be chaotic although it is planned development where the emphasis is on quick software development and not the documentation, even though making the documentation is also a part of agile software development methodology. The agile software development has been defined as [Agile Alliance]:

"Individuals and interactions are valued over processes and tools.

Working software is valued over comprehensive documentation.

Customer collaboration is valued over contract negotiation.

Responding to change is valued over following a plan."

5. Iterative Software Development

Software is developed incrementally while using the Iterative software development method. This allows the developer to take advantage of the experience and user feedback that was acquired during the development of the first increment while creating the next deliverable increment of the project. The project starts with the simple implementation of the entire project, which evolves and gets enhanced with each successive increment while working closely with the end-users. This goes on until the full system is implemented and delivered.

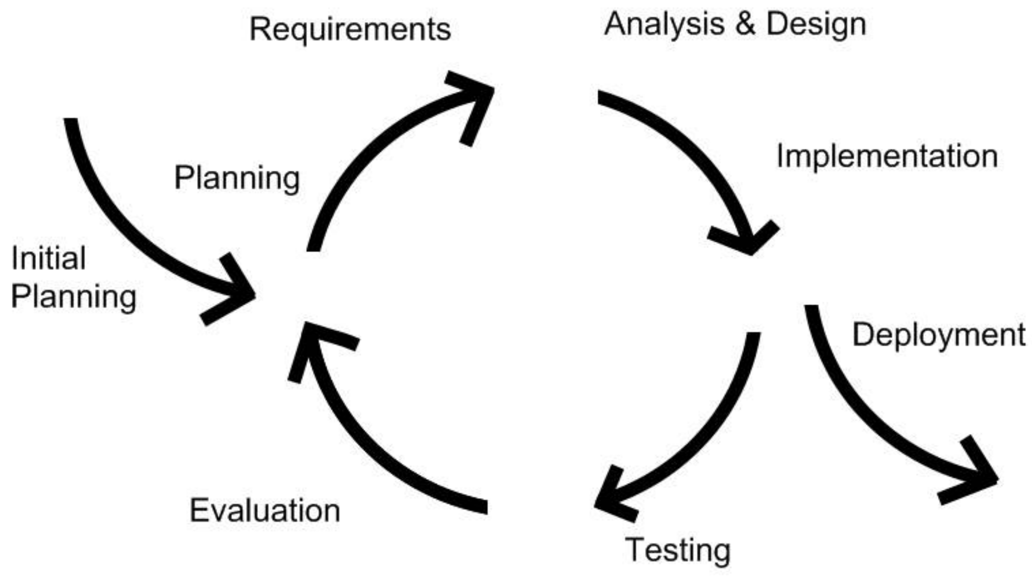

Figure 5 – Iterative Software Life Cycle

The iterative software development consists of the initialization step, the iteration step and the project control list. The initial step is used to create a product that the end user can use and interact with. This provides the development team with the necessary feedback that can then be used to enhance the product in the next increment. In the initial step, the product needs to be simple and should provide a solution to the main problem in question. You can call it as the skeleton of the final project, which performs all the necessary actions. The way the user reacts to it defines how the product is to be created in future increments.

The development phase may contain one or more iterations whose aim is to redesign or to enhance the current state of the project. Several successive iterations are used to design the software according to the client's request.

The project control list keeps track of all the necessary tasks that need to be performed while implementing the next iteration. The new features that are to be implemented and the redesigning of the existing solution are all kept by the project control list. This list is constantly revised and updated during the analysis phase of the project.

The guidelines that define the iterative software development are as following:

- If there is a difficulty in designing, coding and testing a modification, this means that redesigning and recoding is necessary to solve the problem.
- Modifications should be implemented to the modules easily. If that is not possible, then redesigning is in order.
- As the project progresses, the modifications should become easier.
- The existing implementation should be analyzed frequently to see if it matches with the required criteria of the software to be delivered.
- User feedback should be taken seriously and the deficiencies in the project should be overcome as soon as possible for the next round of user input.

When working to create games, constant user feedback is extremely important. Games are meant to entertain its users; hence the main aim for any game developer is to have software that provides both a challenge and entertainment to the user. Games normally go through a long series of iterations and testing before an end product is delivered. This is the reason why the iterative model is extensively used for game development. Now that we know the basics of the iterative software development, we can look into game designing through the iterative software development method. Once that is defined, we will look into the advantages of using the iterative game design and then the disadvantages of the same model. This will bring us to the improvement as suggested by me to the iterative software development method, which puts the user in the developer's seat and helps the users implement the changes without the need to wait for another iteration to be completed.

6. Iterative Game Design

Game design is an iterative process which includes a long a repetitive cycle of designing, prototyping, play testing, and then modifying the underlying design based on the results gathered after each cycle. The iterative game design process holds the premise that a great game cannot be created by simply envisioning it, writing its specifications, and then building it. Through repetitive experiences it has been known that play tests lead to refinements that are critical to tuning the play experience. Prototyping and play testing are a fundamental part of the overall game creation process and they should be started as soon as possible to approach a refined end product.

Iterative design is a design methodology for game development which is based on a cyclic process of prototyping, play testing, analyzing, and finally refining a work in progress. In iterative design, all the interaction with the designed system is used for evolving a project through series of successive versions as they are implemented. An iterative game design can best be described by Figure 6:

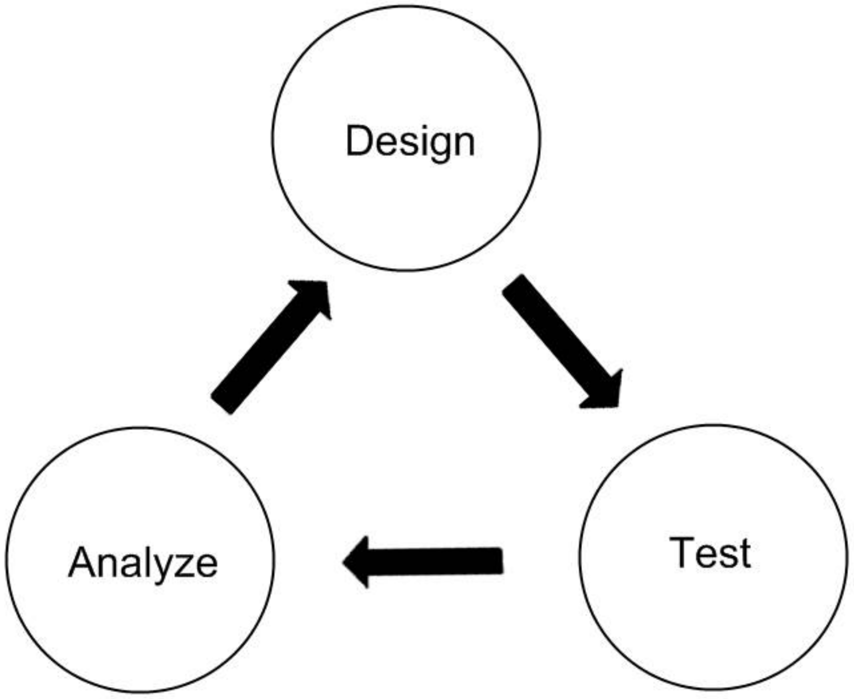

Figure 6 – Iterative Design Method

In an iterative process design, decisions are based on the results and experience acquired from the prototype in progress. The prototype is tested and analyzed, based on which revisions are made, and the project is tested once again. In this way, the project develops through close interaction between the designers and the testing audience. All the feedback received during play testing is forwarded to the designers which after tweaking and adding more features release a newer version. Another important point that is to be considered is that, different users behave differently with the same software. Hence the response from the different users can be varied so a middle ground is sometimes needed to be achieved before the final product is released.

When we talk about iterative designing in games, it normally means extensive play testing. The main emphasis in game development is creating a game that lets its users enjoy and provide a challenge. If a game is too easy, the users will beat it too quickly and if the game is too hard the users will not try to play it for fear of losing every time. Thus, it is only through extensive testing that the game designers can find the middle ground between fun and challenge.

This iterative process which involves extensive testing differs from the typical retail game development. Normally, at the start of the design process, a game designer will think of a finished concept, his entire vision and then write a design document that defines every possible aspect of the game that can be thought of. However, the final game does not come out to be like the original concept. This has a lot to do with what the game designer had built according to his own experience, and what the game testers provided as feedback to the original design. The users play a vital role during the game design process in the iterative game development model. It is important to survey the type of users that will be playing the game instead of creating a game out of developer's experience. The players may belong to several categories and different backgrounds; thus their inclusion in the game design process is critical. Since empowering the users with the tweaking and development task is the emphasis for this project, it is vital that we go through the types of users that normally play the game, once it is released.

Olli Sotamaa, has tried to perceive the type of players that are normally interested in playing a game [Sotamaa]. Players are normally divided into two major categories, the novice players and the professional players. This division is mainly useful for deciding the various degrees of difficulties that must be added to the game. The novice players are looking for a fun and challenging experience and want to beat the game once. The professional players want to go one step ahead and try the game at the hardest of difficulties to really see how good they are with their understanding of the game.

The other division that is normally made is between the casual player and the hardcore player. The casual players are those who pick up the game to kill time and are not interested in all the secrets that are needed to be a better player; whereas the hardcore players want to learn everything that is there to be a better player than anyone else. This division is primarily for the marketing purposes so that the players understand what sort of a challenge they may come across when playing the game and what game best suits their playing style.

6.1. Players as Game Designers

It has been known that game designers normally place cheats inside the game which changes the game itself. The concept of the cheats is to provide a different set of rules that may help the players in beating the game easily. Whenever a player changes the rules by using the cheats, he does in fact take part in the game design process. He tells the developers that this is how he would like the game to be and this is how he would like to finish the game. This concept has been so popular that almost every game that comes out has its own set of cheats. Another interesting fact is that sometimes these cheats are put in the game for demonstration purposes and to test the game. Both of these functions cannot be done unless the presence of cheats was not taken into consideration while creating the game. Rollings and Adams believe the changing the rules as actually an act of game designing as they have said, "Every game player is a potential game designer" [Rollings et al., 2003] .

Recently, Massively Multiplayer games have been very popular with the gaming audience. An important point to this trend is the ability of players to tell their own stories. Players are given a vast world to explore, alliances to make or break and do many of the every day tasks within the game. This provides them the flexibility to change the game and as a result, all the game designers for the current Massively Multiplayer games in development are advised to create a game that changes according to the actions of the player.

Another term that has been in recent use is "Modifications" or mod for short. Games these days come with their own Software Developers Kit, which contains just enough code to let the eager end-users to change the game according to their liking. Previously, it was not appreciated that the original game content should be made available to the end-user for making changes, but recently, it has been found that this increases the life expectancy of the game as the game engine is constantly under use by eager developers. The same game may be changed to look entirely different than how it was originally envisioned and sometimes, it is just polished by the end-users into something better. This also provides the software industry with the potential developers for hiring and minimizes the amount of effort that is taken during the hiring process. Salen and Zimmerman have defined this ability of the end-users to change the game content as, *"one of the sweetest pleasures as a game designer is seeing your game played in ways that you did not anticipate"* [Zimmerman et al., 2003].

The game design process can be distributed into four stages, concept design, pre-production, production and post-production stages [Fullerton et al., 2004]. It is a general belief that the game designers can get some clear benefits by using different user-centered design techniques during the abovementioned stages [Sykes et al., 2006]. Of all the design methods that are used for game development, the iterative design method is considered to be the best as it relies on user feedback early on during the development process. The game designers are encouraged to create a playable prototype so that user response is recorded during the early stages of development. It is however seen that even though the iterative development provides a lot of benefits for game designers, it is still not the popular approach while making games [Zimmerman et al., 2003].

6.2. Relations between Designer and Player

Olli Sotamaa has defined the possible relationships that can be found between a game designer and the end-user [Sotamaa, 2007]. These relationships reflect on the various design ideologies and traditions that are in use these days. These should be discussed to understand the role of the users in producing a high quality game.

6.2.1. Designer as Player

The first relation is that of a designer with himself as a user. In order to create a game, a designer needs to play a lot of games himself. This allows the designer to understand how other game manufacturers have gone about creating similar games and helps him chalk out many design decisions. One of the drawbacks of this is that, the designer might end up designing the game for himself as he himself is testing it as a player. This results in creating games which are mediocre at best. Playing the games by the designers is a good starting point for making any game but after that continuous input by the end-users is essential for making a good game.

6.2.2. Player as Designer's Muse

Sometimes, the designer uses the ideas given by the players for developing new features into a game. This leads to many new and innovative ideas that the designer has not thought of himself. The downside to this is that many of the ideas might not be according to the overall theme of the whole game idea and might work as unnecessary bells and whistles rather than anything that changes the whole gaming playability.

6.2.3. Player as Designer's Patient

Many times, it has been found that the interface scheme that was used for the game was too complex for an ordinary player to understand and use effectively. This causes a large audience not to pick up and play the game and thus results in a complete failure of the game in the market. This means that the player should have such sort of a relationship with the designer as a patient has with the doctor. The patient comes up with a problem, the doctor diagnoses it and then provides a solution that might help the patient. Thus new design decisions are taken by the designers to cure the problems that the players are having while playing the game's prototype.

6.2.4. Player as a Designer's Advisor

Working with focus groups provides a good conception about the overall game design. The central and most effective method for getting players' advice is through playtesting. The proponents of iterative game design agree that the most important time for getting player feedback is during the early stages of game development. Even if the designer to has a good understanding of the focused group it is still not possible to assume how the user feedback is going to be as the designer will never know in how many ways his game will be played and how to assume that at what point the player is going to do what. The open ended nature of the game and the user interaction is what makes the game development one of the toughest jobs in the software development industry.

6.2.5. Players as Designer

There are signs that some developers are considering opening parts of the early game production pipeline to players for their input [Banks, 2005]. This has a lot to do with the success of the player developed content for many game titles, which in turn, has boosted the sales for that game. The two modifications that have become extremely popular are Counterstrike modification for Half Life and Defence of the Acients modification for Warcraft 3. Thus the trend has now become to hire players as the designers for upcoming titles directly from the players community.

Thus the modern trend in iterative game development method is to involve the users from the early stages of the game designing process so that the cost of making design changes can be reduced early on thus maintaining the survivability of the game through the entire development process and after release. A lot of times it has been seen that the remake of the old formula is presented to the gamers time and again before experimenting with something totally new. The designers try new ideas only after they have perfected the old ones and are sure to venture the new territory with as much careful planning as possible. Many other developers mimic the ideas that had been done by other designers which lets them cash on the success of others. This again goes under the umbrella of those designers, who are players themselves. They have tested a popular formula, they know its pros and cons, hence making a game on that idea minimises the risk of failure.

But in all the iterative game development models the problem that has been the most prevalent is the number of iterations it takes till the final product is released. A big reason for this is the amount of recompiling and testing that goes on before the final product is released. The software, Gamedit, as created by me, provides a game design idea that minimises the number of iterations that are needed for tweaking a game. Apart from that, it also provides the players or the testers with editing capabilities, which removes their dependence on the coders for changes to be implemented. The game comes with an easy to use editor which helps in making changes to all the objects inside the game. Once the changes have been made, these are saved in script files that override the initial settings for all the objects. Since the application also reads these script files during run-time, thus no recompilation is needed for these files to work.

In order to better explain this improvement done to the traditional iterative game development method, we need to go through the stages of development, discuss the design decisions taken and finally explain how the editing done by the users impacts the overall software. This will be explained by showing a game and then letting the user make changes to it in order to see how it changes the overall game.

7. Stages of Development

Gamedit went through a lot of iterations before settling into the shape that it is now. It started with a concept of a game where the objects can be defined by the user. Thus any object that is developed by the user becomes the part of the game environment. This was a good concept but it did not provide the user the ability to change the software itself. This idea only gave the user the liberty to add extra content to the already complete software. The two inspirations that I mentioned before (Phun and Crayon Physics Deluxe) do the same as they let users add extra content to the already complete game engine.

The original idea was changed and then a new idea took shape which allowed the users to change every aspect of the game itself, starting from the local menus and going all the way up to the each individual game object. Functionality was put in so that the user will be able to tinker with the game engine itself, hence changing the playability of the game from inside the game.

Apart from that there was functionality built into the game engine itself that let the users define animations by performing them themselves. The idea here was that each action done by the user will be stored by the software itself and then replayed by mimicking the user inputs and redrawing all the performed actions.

Another idea that was linked with the animation definition functionality was to define an easy to use programming language that guided the user to select the next series of steps done in the animation. This was meant to provide the users with more control on how they were going to control the whole animation. The two animation ideas, however good, required a lot of effort to be put into a topic that was loosely connected with animation at all. Here the idea was to develop software that helps in rewriting itself. Software that over-rides the hard code defined in it in favor of what the user wants to add to the software.

Making every feature of the software editable was a hard thing to do in itself, hence spending more time in defining animations not only took the software out of its proposed aim but also created a whole set of new problems. The goal was to provide a software that lets the user define its own art, look and graphics and can be changed in runtime so that the changes can be seen almost instantly. Completing this feature was a priority and it took preference above all the rest.

8. Application Design

Gamedit aims to make the entire software editable. Every menu in the software, that relates to the game itself; every object that is contained in the game and all the attributes within each object should be made editable. This is no small task as each object might be made up of many sub-objects, which in themselves might be made up of many other sub-objects. Thus a long hierarchy of objects is maintained all the time and going through this hierarchy, making changes and putting everything together is no small task. To make it work properly a clever design strategy was thought out and implemented which makes the handling each individual object easy.

It should be made clear that even though this software aims to make everything editable, there are some parts which have been left out of the whole editing process. These parts were related with the in-game editor that is used to change the game objects. Hence, all the menus that can be accessed through the "Options" menu can not be changed. Similarly, the "Options" menu itself cannot be changed. Another reason for doing the same was due to the reason that these parts of the software are not part of the actual game; hence changing them holds no importance on the overall game design process. If editing of these parts had been made the part of the overall game design, it could have complicated the user interface and once the software had been changed from the original state, it would have been very difficult to take the software back to a state where it was in the beginning. These parts have been hard coded into the program and are not subject to change.

8.1. Distributing Objects

The first phase in object distribution was to separate out the main parts of Gamedit which make everything to work together. This required distributing the objects into Menus and Game Objects.

8.1.1. Menu Object

The objects for menus will be made up of components such as buttons and a background object, to draw the menus properly. These objects will themselves be made up of other objects which can be edited to change the shape of each individual button and background. Thus, the hierarchy for every menu can be best described by Figure 1.

Every menu is made up of the MainMenuObject class. In order to show the menu properly, it is then further distributed into two components, the MenuBackground and the Button Component. Both the MenuBackground Class and the Button Class inherit from the Component Class. The MenuBackground is used to display the background of the menu while the Button class is used to draw the buttons for the menu. The Component class contains a list of all the sub-objects defined by the Shape class. These sub-objects are used to draw the shape and colour of the menu and the background.

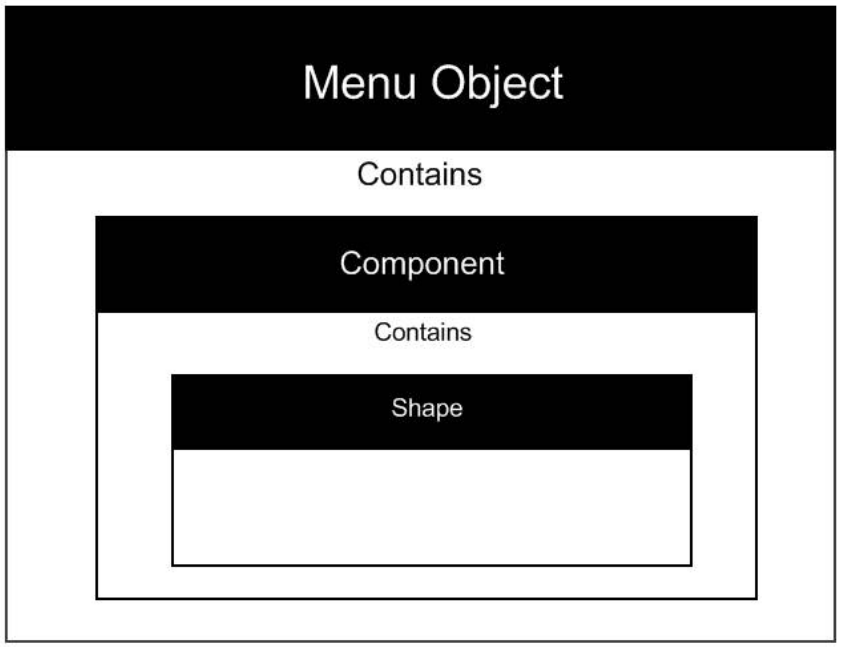

Figure 7 – Hierarchy inside an object of a menu

8.1.2. Game Object

The hierarchy defining the game objects is the same as that for the menu object. The difference here is that the classes that inherit from the Component class are slightly different in functionality than those that are used while defining the Buttons. However, the shapes that are used in defining each component are the same as the same editor is used to alter all the components. The hierarchy inside the object of a game is best defined in Figure 8.

It should be noted that however the hierarchy for both the menu objects and the game objects is the same, both perform totally different functions when it comes to defining the Gamedit program. Even though the user might be able to change the look of each individual button of every menu, there are a few things that cannot be changed, such as the number of buttons in a menu. The creation of these buttons is dependent on the number of objects or sub-objects for which the menu is being created.

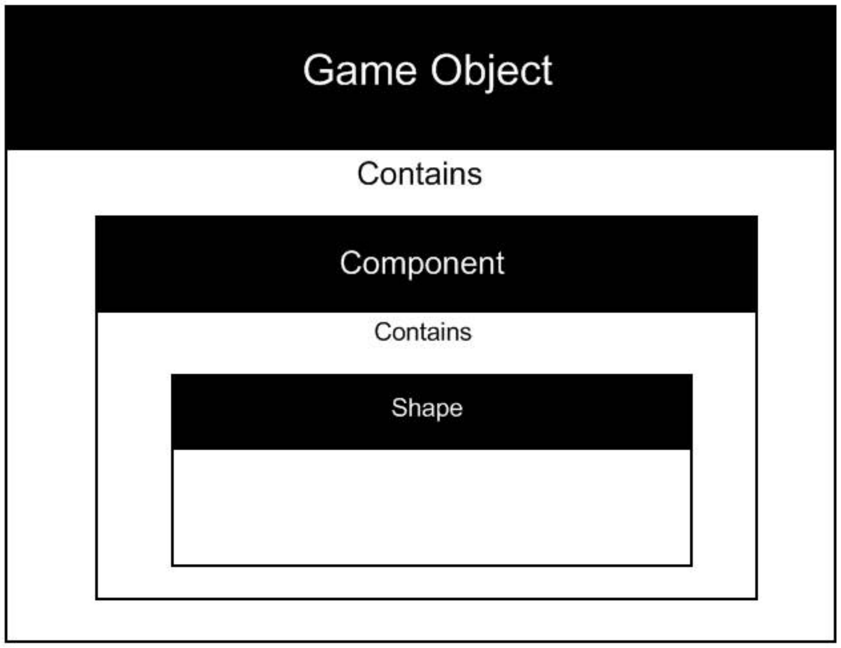

Figure 8 – Hierarchy inside an object of a game

Thus, from the main menu, if the user chooses to open the editor, he is first given a list of menu and game objects that are present in Gamedit. Once he makes his selection, he is then given a list of sub-objects of which the main object is composed of. So once the sub-object is selected, the editor is launched and the user can manipulate the selected sub-objects. Whatever changes the user makes are applied instantly to the selected sub-object which in turn, changes the look of the main object.

8.1.3. Component

Component class is the workhorse for all the main objects of Gamedit. All the major objects are made up of components and it is the combination of these components that we get a completely defined menu or game object. It is by editing these components that we are able to change the main objects in Gamedit. This can be done by adding, removing or changing the shapes that define every component. Once one of the main objects has been selected for editing, the sun-object selection menu comes up to let the users alter each individual component. The components behave differently when they are a part of different main objects, but on the whole, their aim is the same and that being defining the one of the main objects.

8.1.4. Shape

Every component is made up of many different shapes. Shapes are made up of rectangles, triangles, circles and semi-circles. Every shape can be defined in a different colour and the combination of these shapes goes on to define every component. These is no limit as to how many shapes can be in a component, hence the user can define as many shapes as he likes to define a component.

9. Implementation

In order to understand how all of these objects come together, we need to take a walkthrough from the beginning to the editor and then see how the changes done by the user while using the editor changes the software itself. But before we go into the working of the software, we should at first talk about the environment that was used to explain the idea behind improving the iterative game development model.

Java Micro Edition for mobiles was built specifically for portable devices that have come into normal use these days. With an increase in the number of mobile devices, there was a need to have a language that could create small applications quickly and with the least amount of memory footprint. Apart from that, the portable nature of Java, which allows it to run on almost every device, was another reason to be chosen for portable devices.

It has also been seen that during the times when the users are travelling or waiting for an appointment, they tend to use their portable devices for entertainment. This also created a huge market for game software for mobile phones. J2ME is an effective language for fast mobile software development as it allows creation of professional looking games in less than three months' time. Three months for creating a game is extremely fast for any game development which makes J2ME a favourite choice for creating games.

Irrespective of which programming language to choose, the software development model remains the same. This means that no matter what language is being used for developing software, the methods that are going to be used to go about its development are going to stay the same. Since J2ME helps in developing software much quicker than any available language at the time, it was my choice for trying the improved iterative software development model. This helped in creating Gamedit is much lesser time than it would have taken, had I used any other programming language. Also, the emphasis here was to have complete software to see how the suggested improvements worked rather than how beautiful the over all software looked. Thus the first phase, or shall I say the initialisation step during the development of Gamedit was to have a game engine and an editor implemented to see how the basic editing tasks worked on the overall game. Thus, to explain the improvement of the iterative game development model, I shall describe the development of Gamedit through every iteration to explain how this software speeds up the entire iterative software development procedure.

9.1. Defining the Game:

In order to explain the iterative game development model, an elaborate game was not needed. The aim here was to make a simple game that was easy to make and had the least amount of rules so that the editing the game was easy plus the users might not lose track of all the changes once they had been implemented. Thus it was decided that a simple 2D car racing game will be used. The car will be one of the game objects and there will be obstacles in the game that the player will need to avoid in order to win the game. Obstacles will also be game objects and they will also be editable. The basic idea of the game can be described by Figure - 9. This is not the final state of the game and should only be used for demonstration purposes.

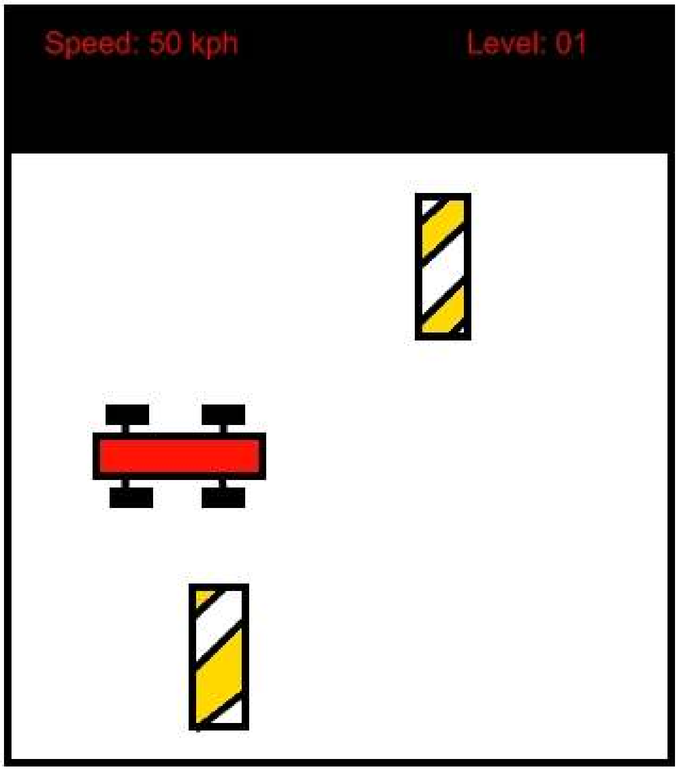

Figure 9 – The Game Concept

In order to define all the game objects, it was decided that all the objects should be distributed into sub-objects. It is only by combining these sub-objects that we are able to define a complete object. This decision was taken because it is possible that we might need to define an object with moving components. If that becomes the case, it will become problematic to define the animation of the object as the whole object will need to be animated and will have separate painting functions or separate functions for defining the different frames of animation.

Going by the natural order of how any object is composed in real life, we normally see a combination of sub-objects which when combined together form an object. This also provided a modular approach for defining the game objects. Now instead of changing an entire object, the user can make a change to one single sub-object. So for example, the car in the game can be divided into the car body and the four wheels. Now each tire can be changed individually and that change will be displayed when the car is drawn again while playing the game. Thus, the hierarchy of objects which defines a car can best be described by Figure 10:

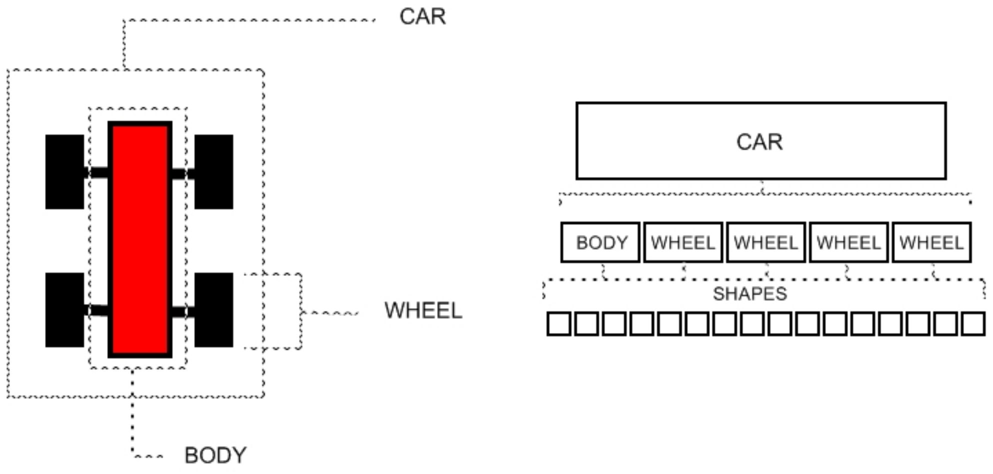

Figure 10 – The Car Object

From this figure, we can easily deduce the hierarchy of the car object. The car is basically composed of five main components. One is the body and the others are the wheels. Since all the wheels can be defined separately, we are considering them as four separate components. Like all the real world objects, these components have shapes of their own. Hence, it is the combination of these shapes that a component is defined graphically. What sort of a shape is used, where it is used, what was the size of the shape and what was the colour, etc, all these properties of every shape are remembered and stored within each component. Thus, when an object is ordered to be drawn, it actually draws all the components of which it is made of. These components in turn call the drawing function of all the constituent shapes and thus the entire object is displayed. So if a user wants to change the look of an object, he needs to go down to the level of the sub-objects and change the shapes that are being used to draw the individual components.

The same holds true for all other objects in the game and also for the menus. The menus themselves are objects and they are also made up of different sub-objects. So if a user wants to make changes to a menu, he needs to open the editor and then select the appropriate menu to change. Then he will be given a list of the background and the buttons that the user wants to change. He can pick up any sub-object and make changes as he pleases. In this way, the entire game can be changed by the user.

9.2. Creating the Editor

There were a few requirements for the game editor to fulfil. First of all, it should be run able from inside the game engine. Then it must be able to display the sub-objects of all the objects, help in editing each sub-object, provide different shapes to help with drawing the sub-objects and finally maintain the history of all the steps done while drawing .

The role of the editor is to change the look of all the sub-objects within the objects and once these changes are done, they are applied to the game without the need of recompiling. This removes the communication cycle that normally takes place between the game coder and the game tester where the tester tells the coder of the many features he wants to be changed and then the coder open up the code, recompiles it and then gives the tester a new build to play on. Now the tester can simply open the in-game editor and make changes as he sees fit.

The menu provides the user with a colour menu so that the user can choose what colour of the shapes should be that he is going to add to the component. He is also provided with a menu to select the type of shape that he is going to add to the component. He is also given the option to resize the shape that has been added and finally, he has the option to undo any move he makes. It should be kept in mind that all the graphics that are used are vector in nature. This ensures that the changes done in the code remain the same irrespective of the screen size of the mobile device. A simple algorithm can be written to resize the shapes as according to the screen on which they are being displayed. Thus, no images are kept in the software itself, which makes the size of the software very small and reduces the amount of memory that is required to run it.

9.3. Creating the Menus

Almost all of the menus in Gamedit are dynamically created. As I had mentioned before that the aim was to create software that is totally editable, it was necessary that all the menus needed to be editable themselves. Every menu is an object itself and a separate canvas has been dedicated for displaying the assigned menu. Apart from the main menu, all the menus depend on the objects and the sub-objects present in the game. Since the main menu needs to specify the main functions of the game, the options are hard coded as a list. But even here, if you increase the size of the list, the number of buttons will increase as well.

Every menu has a Tool tip at the bottom to tell the user at what level he is inside the menus. The Objects Menu lists all the main objects that are present in the game. Once an object has been selected, the Sub-object Menu comes up which displays all the sub-objects that are present in the selected object. Once a sub-object has been selected it is then displayed in the editor and can be edited there.

9.4. Walkthrough

The improved iterative software model for game development requires that the game engine is at its place before the entire editing and tweaking begins. Once a game loop is in place, all that is left is putting in the game rules to define the game. For defining the racing car game as I have discussed above, the first step was to create the game engine and make all the objects editable while using the menu. Right at that time, there was no definition of the game's rules in place. So we start our walkthrough of out project from that point onwards.

9.4.1. The Main Menu Screen

The main menu screen contains the main menu and it is used to make the necessary choices for playing the game or editing the game objects. Figure 11 displays all the menu options that a user can select. If the user selects ”Play” he can play the game without changing any of the game's objects. If the user selects Editor, then he is first asked about the object that he wants to edit and then gets to edit that object. Finally, If the user want to exit the software, he can just select the ”Quit” option and the software will stop execution.

It should be noted that although the text of the buttons for the main menu were hard coded, the creation of the buttons was not. If the text is too long, then the buttons extend themselves to cover the entire text, depending on the size of the text.

The user can navigate the menu by pressing the arrow keys and then press the middle soft key to make a selection. The soft key for menu selection can change depending on the phone it is being used. Sometimes the number keys might be used to move around the menu and making a selection.

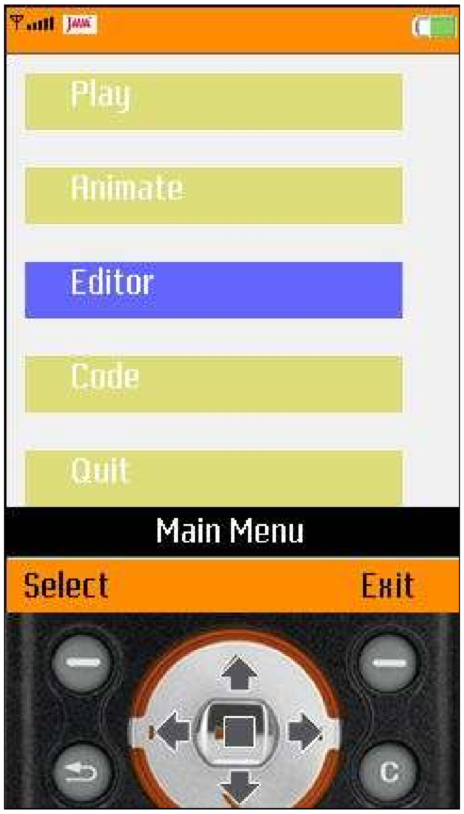

Figure 11 – The Main Menu

9.4.2. The Object Menu Screen

Gamedit contains a linked list which stores all the objects that are contained in it. This linked list is required to create the menu for all the objects. The Object Menu comes up when the user chooses the editor as his option from the main menu. Gamedit accesess the list of objects that are stored in it and reproduces those objects as selectable buttons for the main menu. All the objects have a String attribute called ”type” which stores the name of all the objects. This holds true for all the sub-objects as well. This is necessary so that we can identify each object individually. It is possible that many different objects might have been created from the same class, so if we use the class’s name to define every object that will cause confusion. Thus whenever the menu is created, the type attribute of all the objects is read and is put as the label for the button defining that object. Figure 12 displays the Object Menu.

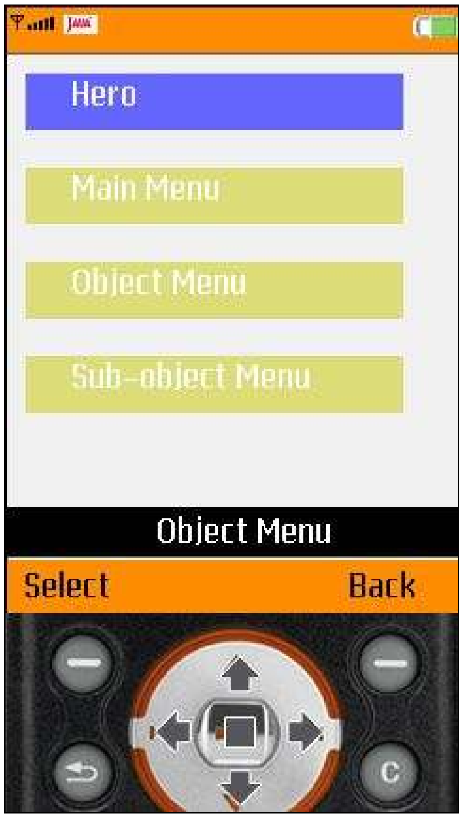

Figure 12 – The Object Menu

In the screenshot above, we can see that in the Object Menu, the game objects and all the menu objects are displayed. The user can select any of the given menus above and that will lead him to the sub-objects. Here we can see that only four objects are being displayed. This is all dependent on the number of objects that are present in the game. As we increase the number of objects, the number of selectable options in the object menu will increase. Selecting an object from the object menu leads the user to the sub-object menu.

9.4.3. The Sub-object Menu

The Sub-object menu shows all the components of which the selected object is made up of. All the components themselves have the ”type” attribute, which helps in labeling the selectable buttons of the menu.

The number of buttons that are shown in the menu depend on the number of components that are present in the object. If the components are the buttons of the menu itself, then they are shown as they are shown on the menu screen as shown in Figure 13. The user can add more shapes to these components and make them look different.

If the user chooses to change the shape of one of the components of the game objects, then only that component of the entire object is displayed in the editor.

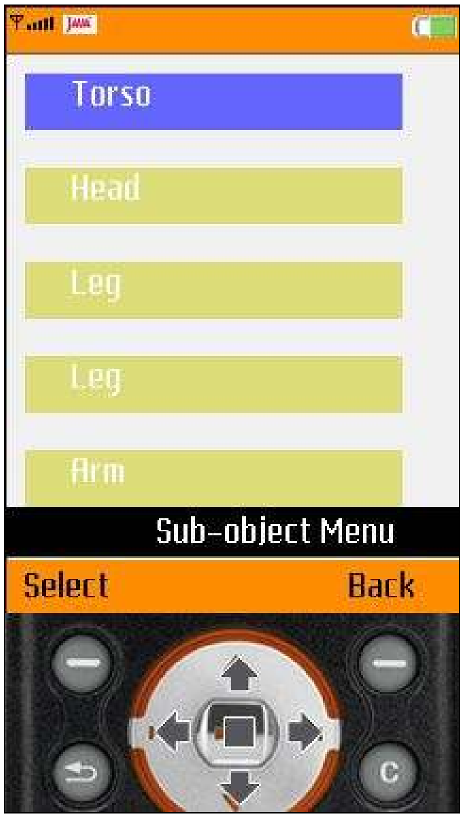

Figure 13 –The Sub-object Menu

9.4.4. The Editor

Once the user has selected the sub-object to change, then the editor is displayed with a red crosshair in the centre along with the component that is to be altered as shown in Figure 14. The cross hair shows the position where the user is and helps in pinpointing the place where any new shape is to be added.

The crosshair can be moved around using the arrow keys and the middle soft key is used for making a selection. Here, we can see that the user has selected the Torso of the Hero object from the sub-object menu. Any changes made to the Torso component will show up when the user will play the game.

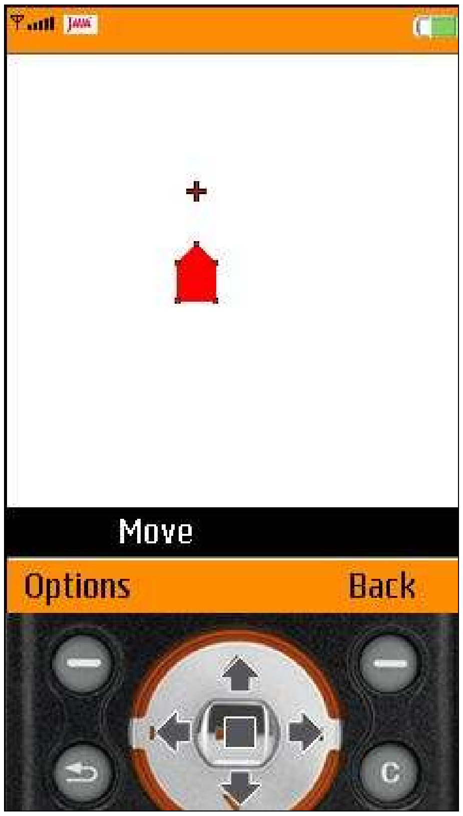

Figure 14 – The Editor

By pressing the left soft key, the user can bring up the Options menu shown in Figure 15. The Options menu can help the user to select a colour for the shape that is going to be added and also to select the shape itself. Apart from this, once an action has been taken, the Options menu has the “Undo” option that lets the user undo an action and finally, it can be used to return to the main menu.

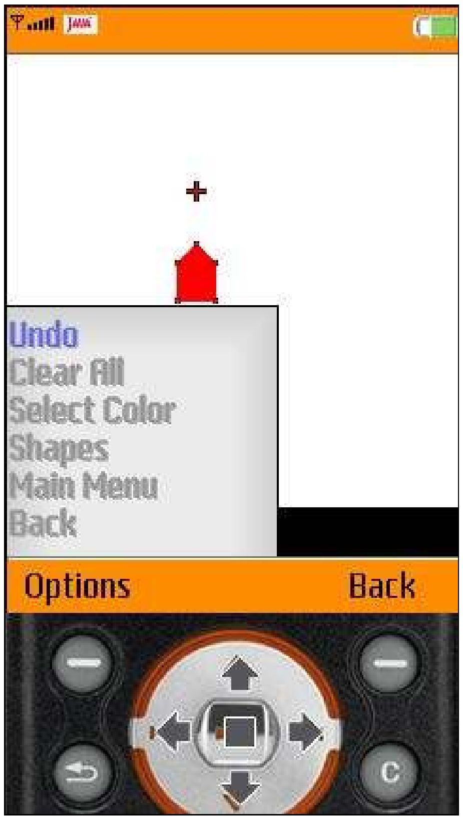

Figure 15 – Options Menu

If the user chooses to select the colour, then Colour Menu pops up. Here, the user is given a grid of colours which can be navigated by using the arrow keys as shown in Figure 16. Once the user has selected which colour to select, he can press the middle soft key on the phone and that colour will be selected.

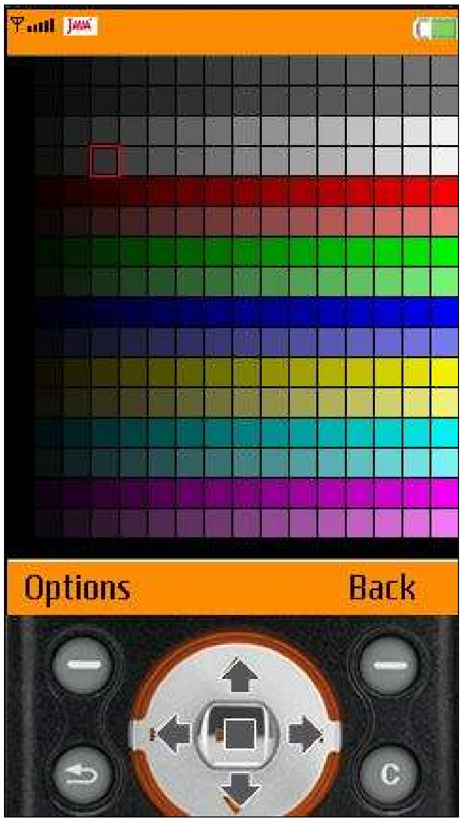

Figure 16 – Colour Menu

There is also a Shapes menu present (Figure 16), which can be selected by choosing "Shapes" from the Options Menu. This gives a list of shapes that the user can add to the component. If the user has selected a colour before selecting the shape, then the selected shape will have the selected colour. If the user selects a colour after selecting the shape, then the selected shape will change its colour to that which was selected. It should be kept in mind that a shape can be placed anywhere on the drawing canvas. Hence it is possible that while drawing the user might add shapes randomly. Even though this feature is supported by the editor it is going to be aesthetically unpleasing to the user to see disjoint shapes to represent the components. The best practise would be to think before drawing any component and be careful while adding shapes so that the main object does not look bad once all the components are drawn.

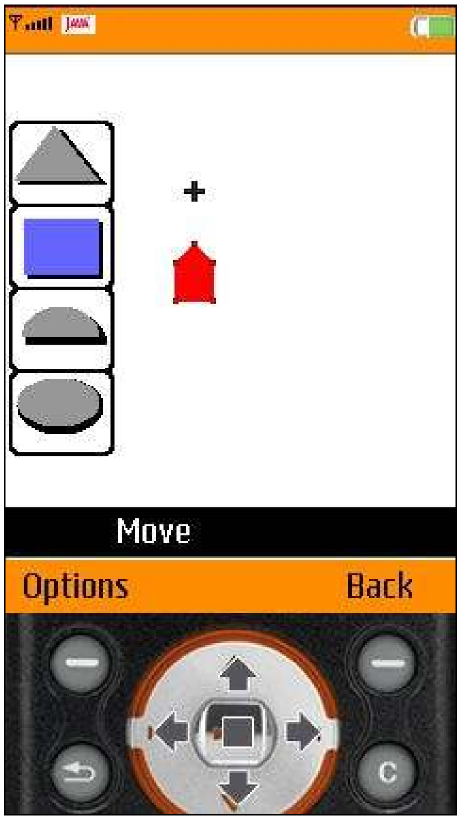

Figure 17 – Shapes Menu

Once the user is happy with his choice of shape selection, colour usage and overall drawing of the component, he can simply select back from the right soft key, go back to the main menu and select the "Play" option to play the game. Now the same game will be displayed exactly like how the user drew all the components. If the user wants to tweak any of the components, he can go back to the main menu and start editing again.

This is the most important feature of this thesis and that being, giving the tester the power which the game programmers normally have. The reliance of the testers on coders to play an updated version of the game has been removed. The changes that they want can be applied almost instantly and they can test the game as they want. This reduces the number of iterations that are needed to tweak the game as that part is done by the testers readily. All the updates are recorded in the XML file of each component and the coders can easily use those XML files to see what were the changes done that made the tester like the game. These changes can then be hard coded in the game if the coders do not want to rely on file reading or they can just let them be in the game to reduce the amount of coding done to tweak the game. Hence, the main concept of this improvement on the iterative development method is to empower the testers with the power of coding and let the testers call the shots for better game playability.

10. Results

This section explains the targets that Gamedit wanted to achieve, the problems encountered and the actual features that were implemented.

10.1. Aims

Gamedit was supposed to provide a tool that had a game engine and an editor together in one application. Whatever objects that were contained in the game, were supposed to be editable by the editor. This aim was achieved by the application as it provided the user with the ability to select each object within the game and edit it. No exception was made whether the object was a part of the game itself or the menu; even the backgrounds for the different in-game screens were editable.

The second aim was to see the changes without having to recompile the code. To achieve this aim, the entire game engine was developed with dynamic objects in mind. The idea was to make it possible to edit the game from inside the game. All the objects in the game were kept in a linked list. The menus were created out of these lists which were then used to select each individual object. Every object was then made up of several sub-objects which were then composed of more sub-objects. The real challenge was to maintain this hierarchy and still make it possible to edit each individual sub-object. These changes then reflected on the entire object and made it possible to change the objects by changing the sub-objects.

The third aim was to create a game with pre-defined rules. The software did not hold much value unless there was a game that can actually be played. Once a game was in place, it was up to the testers to tweak the different settings and make the game enjoyable.

The final aim that was to be achieved was reading and writing of XML files. The maintenance of the XML files was the centre point for this project. Whatever changes were made to the objects were stored in each object's XML file. Since the software was supposed to read the XML files before drawing any of the objects, this ensured that the objects were drawn correctly even if we do not recompile the code. All the XML data was first stored in string with in the object and then written to the XML files. This was the only aim that the project was not able to achieve.

10.2. Problems

As discussed above, the only aim that was not achieved during the creation of the project was the reading and writing of the XML files. This had to do with the emulators not having the permission to read or write files. Since they did not get any permission, writing of the XML data did not take place. This meant that as long as the program is running, the XML data will be stored in memory and all the changes will be reproduced as they are stored in the XML string of every object. But as soon as the program is closed, these changes will be lost.

A work around to this problem was to output all the changes to the console of the IDE. Then the programmer can use the data received in the console to apply the changes. This however limits the users to stay in the vicinity of the IDE to make the changes and using a mobile phone to do the same is not possible. Almost three weeks were spent to find the proper solution to the problem also on different computers but that did not result in much success, so it was decided that all other parts of the software must be completed before returning back to this problem and fixing it.

10.3. Future Enhancements

Talking about future enhancements, the first and the foremost that comes to mind was to fix the problem concerning the reading and writing of XML files. This is a configuration problem rather than a coding problem. All the code regarding the reading and writing of the files is in place. Once that configuration problem is fixed, the program will be ready to be made portable to mobile devices.

Another enhancement that can be made to the current software is the ability for the user to define animations. There is code available within the software that records all the steps done by the user and replays them when needed. In this way whatever animation the user wants to produce, he can define it by moving objects using the arrow keys. More functionality needs to be added, to let the user add and delete objects as the user pleases. The in game editor can also help in defining new objects.

The in-game editor provides only the most basic of functions when it comes to drawing. Many more shapes can be added to the editor to allow the user a lot of options to choose from while defining a shape of an object. Another enhancement that can be added is the ability to rotate the shapes. At the moment the shapes that are added can have their colour changed and they can be resized. The more options there are for more shapes and transformations, the better the overall drawing experience is going to be.

Lastly, in order to support easy to exchange XML files, it will be best if the user can exchange the XML files through MMS messaging. That way the users will not need to open the contents of the mobile phone with a personal computer and will be able to pass contents with their phone only. This increase the accessibility of the combined development and sharing of resources and help in rapid game development.

11. Discussion

Gamedit aims to further enhance the iterative game development model by providing the testers with tools to change the software. It narrows the gap between software analysis and testing by removing the need for recompiling after every test. Software made using this design method must meet some requirements before it can start to take advantage of this approach.

The first and the foremost requirement is that, this technique is only good enough for game development. Even though making games is one of the most difficult jobs in the software industry, the improved iterative model can only be worked on this category of software. No other software requires play testing and careful balancing of all the attributes which may change the game's playability.

The second requirement being that the game is not too complex to code. It was felt while developing Gamedit by using the improved iterative software development technique was that, it is a combination of two types of software in one package. The game designer is not only making a game, but in fact, he is also making an editor inside the game. This however is not entirely different than what normal game developers do. In order to create game resources, such as object models and game maps, the developers create separate tools for making these resources. Gamedit, on the other hand, adds these tools to the game engine itself and tries to allow using them as easy as possible.

Gamedit is totally based on vector graphics. Thus no external images or files can be used to create any resource. This reduces the amount of detail that may be required in designing different objects for the game. The artist is limited to the software's drawing tool and can not do anything beyond that. It is believed that as the developers start to adopt this method for game development, the abilities of the game engine will improve with time and start to incorporate new details. The games that can be made from the current state of Gamedit are strictly 2D and it never ventures off to explore new 3D worlds. Doing this was necessary because whether the game is in 3D or in 2D the basic concept of a game remains the same. The excessive detail would have made the idea complex and difficult to both code and understand.

The ability of the tester to change the code adds another layer to the whole iterative game development model. It can best be described by figure given below:

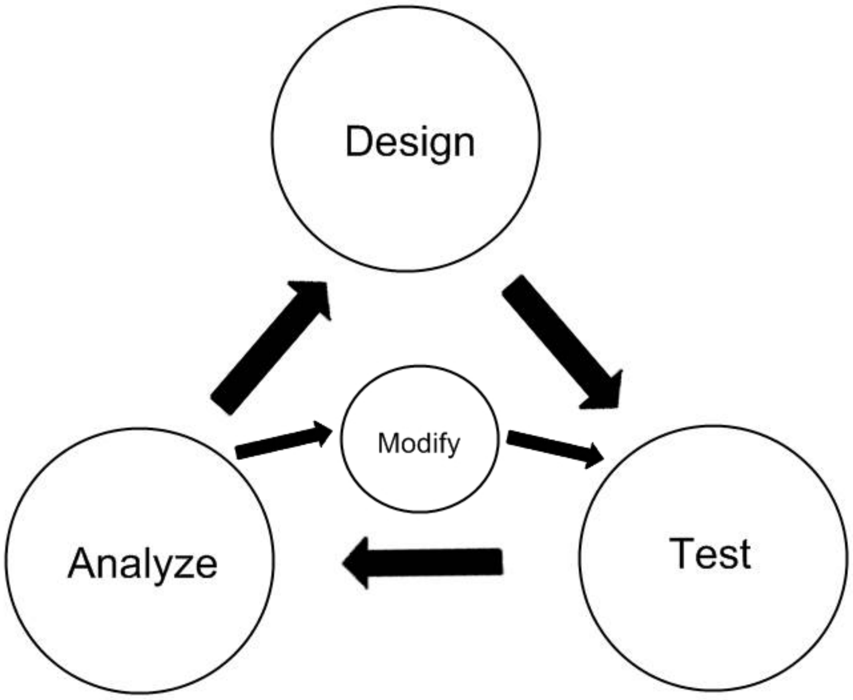

Figure 18 – Improved Iterative Game Design Method

In between the Test phase and the Analyze phase another small phase has been added known as the Modify phase. Thus, in every iterative cycle there is yet another cycle that can be referred to as the tweaking cycle. Software is presented to the user for testing. The user tests the software and makes some recommendations which are then analyzed by both the testers and the coders. If the recommendations made by the testers do not require any major design changes, then either the coders or the testers go about modifying the code using the in-built editor.

This small inner cycle continues to the point where no minor changes are possible. Everything has been tweaked and modified and the changes that are to be done now require an entire design overhaul. In that scenario, the coders take the reigns of game development, redesign the game and then with the creation of the next build they let the testers go on with their Test, Analyze and Modify cycle.

The addition of this small inner cycle keeps the iterative model at heart as the same iterative process is continued but there is an additional layer to it now. This allows for lesser time spent in recompiling and tweaking on the developers end which improves the amount of time taken to develop the game. Apart from this, the same game can be developed by different users by working on separate parts of the project. Once all the XML files have been generated, they can be used to replace the existing files which in turn change the whole game. Similarly, the same game can be given to different artists who then might come up with two totally different looking games.

User involvement in game modification and development is being greatly appreciated these days. Almost every game that comes out nowadays has some developer tools with it. Using those tools is difficult for a user that has no technical background and it is hard for them to start a project by themselves. Gamedit is an experimental framework to both test an improved iterative game development technique and to provide the end-user with easy tools for making their own games. It is believed that if this method is adopted for making small casual games, it can help in creating a large variety of games based on one single game engine which users will not need to buy but in fact distribute for free. This can be then used for teaching purposes and for entertainment.

12. Summary

Gamedit is a software development tool that is based on the iterative game development model. The aim of this tool was to exhibit a new improvement that was made to the old iterative game development model by empowering the testers with the ability to make changes to the software without learning to code. Another important aspect of this tool is the possibility to make changes to the software after compiling has been done. This required that a game engine should be developed that is as dynamic as practically possible and relied on a large number of script files that are generated and rewritten to account for the changes done to the software.

Once the users were given the power for changing the code, it reduced the amount of tweaking, programming and recompiling that the coders needed to do in order to fulfil the demands set by the testers. Testers can now tweak the game themselves and stop relying on the developers unless a major design change is required. This helps reduce the number of iterative cycles needed to complete the game hence speeding up the time required for game development.

Another important aspect is the ability of the end-user to totally change the look and feel of the game once it has been released. It is as easy as changing a few script files from the software and the user will have a totally new game to enjoy. Also, different users can make different parts of the program and then exchange files to complete the whole game modification process. This not only increases the life of the game engine that has been developed, but it can also be used to exhibit modular programming and to learn basic programming thought process and skills.

References

[Agile Alliance] www.agilealliance.org "Manifesto of Software Development"

[Banks, 2005] Banks, J.A.L. (2005) "Opening the Production Pipeline: Unruly Creators" in de Castell S., and Jenson, J. (eds.) Changing Views: Worlds in Play - Selected Papers of the 2005 Digital Games Research Association's Second International Conference, Digital Games Research Association & Simon Fraser University, Vancouver.

[Boehm, 1988] B.W. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, pp. 61-72, May 1988.

[Crayon Physics Deluxe] http://kloonigames.com/crayon/

[Eclipse] http://www.eclipse.org/

[Eclipse ME] http://www.eclipseme.org/

[Ernerfeldt, 2008] Phun beta 3.5 by Emil Ernerfeldt, made for Kenneth Bodin at VRLab, Umeå University. http://www.phun.at/

[Fullerton et al., 2004] Fullerton, T.; C. Swain & S. Hoffman (2004) Game Design Workshop: Designing, Prototyping, and Playtesting Games. CMP Books, San Francisco, New York & Lawrence.

[Goldberg, et al., 1995] A. Goldberg and K.S. Rubin, Succeeding with Objects, Addison-Wesley, 1995.

[Jave ME] http://java.sun.com/javame/index.jsp

[J2ME Polish] http://www.j2mepolish.org

[Martin, 1991] James Martin, Rapid Application Development, Maxwell Macmillan International Edition, 1991.

[Rauterberg, 1992] Matthias Rauterberg - An Iterative-Cyclic Software Process Model, Fourth International Conference on Software Engineering and Knowledge Engineering, 1992, pp. 600-607, Capri, Italy.

[Rollings et al., 2003] Rollings, A. & E. Adams (2003) Andrew Rollings and Ernest Adams on Game Design. New Riders Publishing.

[Zimmerman et al., 2003] Zimmerman, E. & Salen, K. (2003) Rules of Play: Game Design Fundamentals. MIT Press, Cambridge, MA.

[Savani, 1988] Hasan Savani, "IEEE Software Manager Exchange", IEEE Software, July 1989, pp. 108.

[Sotamaa, 2007] Sotamaa, Olli, "Perceptions of Player in Game Design Literature" in Baba (ed.) DIGRA 2007 Conference Proceedings. University of Tokyo, Tokyo,pp. 456-465.

[Sun Java Wireless Toolkit] http://java.sun.com/products/sjwtoolkit/

[Sykes et al., 2006] Sykes, J. and Federoff, M. (2006) ”Player-Centred Game Design”, inCHI Extended Abstracts 2006, pp. 1731-1734.