International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 05 | May - 2022 Impact Factor: 7.185 ISSN: 2582-3930

Car Chase Game: Aegis from Foe

Prof. M.K.Vairalkar
Professor *dept. computer science engineering*
Govindro wanjari college of engineering & technology
Nagpur, India
cvairalkar@gmail.com

Riya Phulewar
dept. computer science engineering
Govindro wanjari college of engineering & technology
Nagpur, India
riyaphulewar1804@gmail.com

Sahil palandurkar
dept. computer science engineering
Govindro wanjari college of engineering & technology
Nagpur, India
alandurkasahil@gmail.com

Ritik Shaniware
dept. computer science engineering
Govindro wanjari college of engineering & technology
Nagpur, India
ritikshaniware15@gmail.com

Prasanna Indurkar
dept. computer science engineering
Govindro wanjari college of engineering & technology
Nagpur, India
prasannaindurkar@gmail.com

Urvesh Rahate
dept. computer science engineering
Govindro wanjari college of engineering & technology
Nagpur, India
urveshrahate@gmail.com

Kunal Shende
dept. computer science engineering
Govindro wanjari college of engineering & technology
Nagpur, India
kunalshende@gmail.com

***Abstract*— This Bachelor thesis describes a case study , where we are focusing on developing a car chasing game , using a process upon agile development; an evolutionary development method.**

The thesis will cover implementation of graphics, physics engine.

In the end , our case study will show that this development process was an appropriate choice for our game development project.

Keywords — software application game development

I. INTRODUCTION

Developing software application is a time-consuming process, and with time-consuming processes come high costs. During the last years, several software development methodologies often known as agile software development, have become widely used by software developers to address this issue .Many different development methodologies can be more or less good , depending of the task and application type. One of the software development methodologies is the evolutionary software method , which as the name hints ,takes on an evolutionary approach to the problem, and allows the project to evolve through different stages of the project. Our case study will show how well this develop a car chasing game .Some requirements for the computer game were given from the beginning, such as

3D graphics – The game must contain 3D models, and render these in the game. 3D environments were never a requirement, and platform games with 2D environment could still open up for 3D objects.

II. PROBLEM STATMENT

The game result must impress whoever plays the game . It should last long and make the players come back and play it over and over again.

Working with these requirements, we decided to use Unity 3D as our platform to challenge for the project group , since all had none or little experience in modelling ,spending time learning how to model proper for our game Working with these requirements, we decided to use Unity 3D as our platform to develop our 3D game with. This decision was made with regard to that the platform had many in-built tools and provided a good framework for us to get started with the development as fast as possible. The fact that Unity 3D also used javascript as development language was also in consideration, since we wanted to learn this newly developed javascript language.

.

III. THE PRPOSED SYSTEM

In this project, we were left free to decide what type of game we wanted to develop. The suggestion was that a racing game would be suitable, since such a game usually do not depend on advanced assets, e.g. animated models. After some brainstorming, it was decided that a racing game should be developed.

Our first step was to collect all the ideas from all the members of the group and if possible discuss and combine them. And the final solution to our question was developing a car chase game in which there will be given time limit and till the completion of the time player has to defend their car from the police cars and if any one of the police car collides with players car then the game will be over. After this we decided to add some exciting levels

IV. MODEL OF SOFTWARE BASED GAME

A software process model

A software process model is a theoretical philosophy that describes the best way of developing software. Based on one or several models, a software process is formed providing guidance on how to operate. A software process model may also be described as an abstract representation of a soft-ware process. The concept of the process model is similar to an abstract java class, which cannot be instantiated, but it can be implemented by another class, thus providing basic guidelines for that other class.

A model may for example demand customer involvement, but it does not state exactly how. A process implementing that model should involve the customer in the process' activities, but is free to choose how.

There is not only one type of process model, but two.

The first one is the most common, and described above. The second type of process model is called a process paradigm, which is a model even more general than an ordinary process model. Such a model does not hold any details on how the activities that lead to the completion of a project should be performed, but what it does hold is basic guidelines of how to develop software, and assumptions about what sort of project could benefit from implementing a particular model. With this in regard, one can conclude that a process paradigm provides a framework that may be adapted to form a process which suits a particular project. There are three major process paradigms that are commonly used today in software engineering practice; the waterfall model, component-based software engineering and evolutionary development.

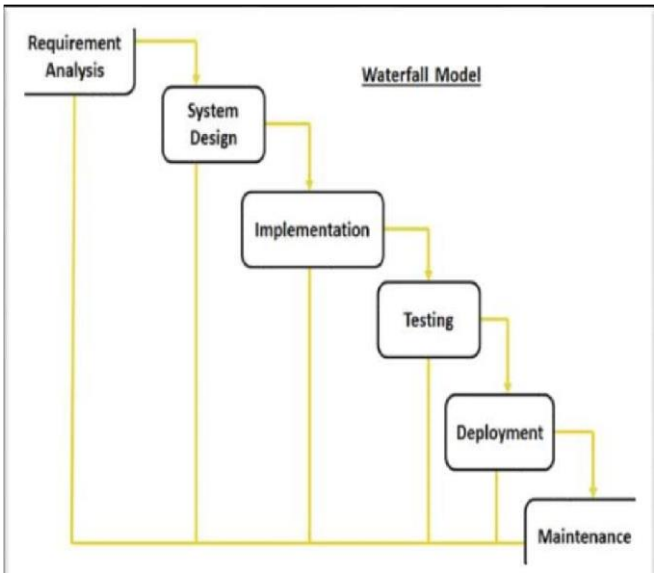
A. The Waterfall Model

The Waterfall Model is recommended for large and complex systems that have a long life-time2. Some systems which carry these attributes are also *critical systems*. This means that if a fault would occur, it may result in:

- Threat to human life (death or injury)
- Economic loss
- Environmental damage

It is believed that the waterfall model would be an appropriate choice when developing a critical system, since the model emphasizes on thoroughness.

The basic concept is to take all the activities and treat them separately. One activity is always followed by another, in the same way water travels down some falls. This description becomes even more obvious when looking at a visualization of the model.



V. ADVANTAGES

1. *Requirements definition:*

All requirements on the system are found by talking to system users. Example of requirements can be services, constraints and goals, such as "We want a webpage that colour-blind people can enjoy".

2. *System and software design:*

In this activity, the overall architecture of the system is established.

3. *Implementation and unit testing:*

The software is implemented in units which also are tested.

4. *Integration and system testing :*

The units are merged together into a complete system. Further testing is required.

5. *Operation and maintenance:*

The system is delivered to the customer and put into operation.

"**Bugs**" are almost always found, and therefore the system required bug-fixing and maintenance.

In each of the activities described above, one or several documents are produced. The idea is not to start on a new activity until the documents belonging to the previous activity is signed off, but in practice, this is not how it is done. Instead most of the activities overlap and all the

documents feed information to the different activities. Although these documents provide a good overlook

VI. ALGORITHM

A. We create two lists – Open List and Closed List

I. Initialize the open list

II. Initialize the closed list put the starting node on the open list (you can leave its **f** at zero) III. while the open list is not empty

1. *find the node with the least **f** on the open list, call it "q"*
2. *pop q off the open list*
3. *generate q's 8 successors and set their parents to q*
4. *for each successor*

b. if successor is the goal, stop search

c. else, compute both **g** and **h** for successor successor.**g** = q.**g** + distance between successor

IV. and q successor.**h** = distance from goal to successor (This can be done using many ways,

V. Manhattan, Diagonal and Euclidean Heuristics)

VI. successor.**f** = successor.**g** + successor.**h**

a. if a node with the same position as successor is in the OPEN list which has a lower **f** than successor, skip this successor

VII. iV) if a node with the same position as successor is in the CLOSED list which has a lower **f** than successor, skip this successor

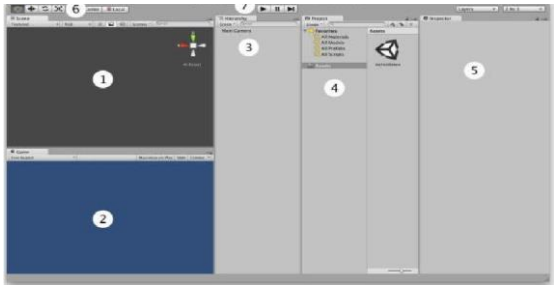
VIII. otherwise, add the node to the open list end (for loop) e) push q on the closed list end (while loop)

*B. **User Interface***

File > New Project

We don't have to import any of the packages at this point, but do specify the save to location in the dialog window. We kept all our project related assets in this location to avoid missing files and broken links later in the game development process.

File > Save Scene as…



User Interface Components:

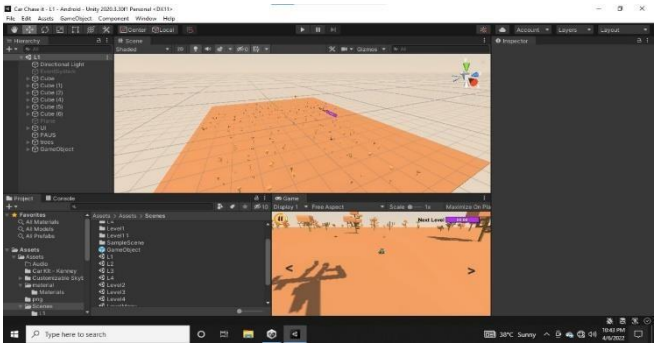
1. **Scene:** This is where you will place any visual assets in your Unity environment. It will update in realtime when you are previewing the game. Note the manipulator on the top right; this allows you to switch between a number of standard views. We are currently in the perspective view (toggle between isometric (2D) and perspective (3D)). Although this doesn't matter too much, it allows us to view our scene with a vanishing point, which is the standard way Unity games will display.
2. **Game:** When you're not actively running the game, it will show a rendering of how the game will look, ignoring graphical effects that need to be computed at run-time, from the point of view of the main camera. When you're previewing the game, you'll be playing through this window. Since our scene is currently empty, all this window is showing is the background color.
3. **. Hierarchy:** This lists all the objects in the currently loaded scene, and any children they may have. **Children** are objects that can be thought of as subordinate to the parent object; wherever the top object moves, they'll follow, keeping the current offset they have to this object. This is an important concept for Unity beginners to understand; we'll cover it more in detail later and in the workshops.
4. **Project/Assets view:** This is a list of all custom assets for our game, including graphical assets, sound, scripts (more on these later), prefabs (pre- assembled game objects), and much more. Our current game is currently using only one empty scene (titled "myFirstScene").
5. **Inspector:** Since we currently don't have any objects selected in the Hierarchy or the Project/Assets view, it's completely blank. The inspector allows us to look at and tweak individual settings of various game objects and assets, as well as adjust some global settings. The Inspector is content-sensitive and changes its parameters based on which game object/asset is selected. This is also a place to show you your project settings and preferences by choosing them from the Edit menu.
6. **Graphical icons for moving the scene and its contents** :The hand allows us to pan around the scene; when combined with other scene camera controls, Unity becomes very easy to navigate (see below). The icon on its right, which looks like four arrows, allows you to move a selected object around. We call this transforming the object. The next icon

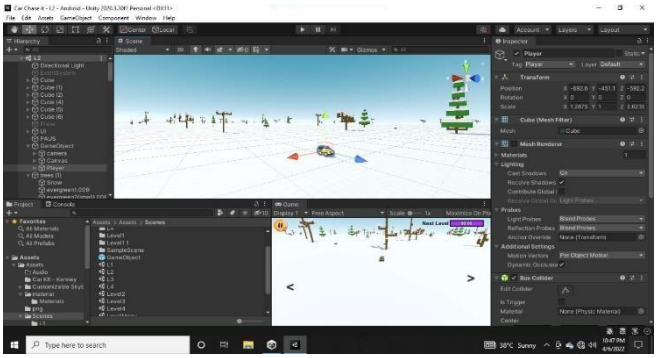
Level 1 – Dessert

allows for rotation of the object, and the final one allows for uniform scaling of the object.

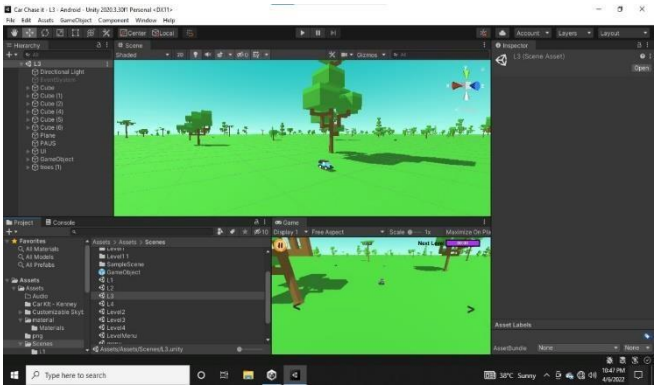
7. **Playback bar.** This allows us to play, pause, and stop running our game in the Unity editor. This is the quickest and easiest way to test and tweak the game.

C. Our game levels





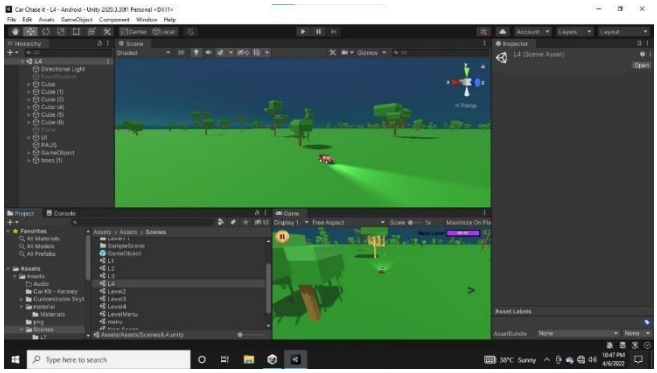
Level 2 – snow Land



Level 3 – Evergreen Forest(Day)

Aegis from foe, a car chase game using Unity Software, and C# language.

➨ The game's physics, camera control, and movement along with background and levels, all of these have been explained in details, with the help of slides



Level 4 _ Evergreen Forest(Night)

VII. Conclusion

VIII. ACKNOWLEDGMENT

We present to you the project paper on **Car Chase Game : Aegis From Foe .**We feel very much delighted in expressing sense of gratitude to our Project guide **Prof.M.K.Vairalkar**, for his timely help during the presentation of the project and for their constant encouragement and valuable guidance. The development of our project would have been impossible without the firm support of our guide.

IX. References

We have gone through certain books, done some research work using internet, understanding the technologies being used in our project work.

The following references are

- **http://unity3d .com/support/documentation/Components/class-Texture2D.html**
- **http://unity3d .com/learn/tutorials/modules**
- **http://unity3d .com/learn**
- **http://catlikecoding.com>Catlike Coding>Unity**
- **http://unity3dstudent.com/**
- **http://cgcookie.com/unity**