Carrom Tutor : Game-based Learning and Implementation

Dissertation

Submitted in partial fulfillment of the requirements
of the degree of

Master of Technology

by

Mrinal Chandra Malick
Roll Number: 123050064

Under the guidance of

Prof. Sridhar Iyer

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai

2014

Dissertation Approval Certificate

Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

The dissertation entitled **"Game Based Carrom Tutor"**, submitted by **Mrinal Chandra Malick** (Roll No: **123050064**) is approved for the degree of **Master of Technology** in **Computer Science and Engineering** from **Indian Institute of Technology, Bombay**.

Prof. Sridhar Iyer
Dept. of CSE, IIT Bombay
Supervisor

Prof. Purushottam Kulkarni
Dept. of CSE, IIT Bombay
Internal Examiner

Prof. Vijay Raisinghani
Associate Dean (MPSTME) &
Professor, NMIMS
External Examiner

Prof. Abhay Karandiakar
Head, Dept. of Electrical Engineering, IIT Bombay
Chairperson

Place: IIT Bombay, Mumbai
Date: 19^{th} June, 2014

Declaration

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Mrinal Chandra Malick

Signature

MRINAL CHANDRA MALICK

Name of Student

123050064

Roll number

19-06-2014

Date

Abstract

Rapid improvement of technology and education increase the necessity of potent tutors day by day. Tutoring can be done utilizing the web or using the traditional method. In this report, we describe the development of a web-based Carrom Tutor, and a game-based Carrorm Tutor implemented in Blender 3D. We refer to these Tutors as Carrom Tutor 1.0 and Carrom Tutor 2.0, respectively. As the title suggests, these Tutors are initiatives for teaching Carrom skills and strategies to the Carrom aspirants. While designing the tutors, Educational Technology, Game Based Learning and Software designing perspectives have been emphasized.

At the beginning of the report, principles of Game-Based Learning and Educational Game Design are discussed. The next chapter is about Carrom Tutor 1.0, it's design, implementation, demonstration and user experiments Then the report covers the motivation behind building Carrom Tutor 2.0, and provides detailed descriptions about Carrom Tutor 2.0's design, implementation, demonstration, user experiments and challenges. Technological tools and platforms considered for implementation, along with details of actual implementation have been described. Finally, the experiments conducted to test the effectiveness of these systems are described, along with possible future works.

Acknowledgements

I thank Prof. Sridhar Iyer for his invaluable guidance. I also thank Mayur Katke for his assistance and cooperation towards achieving the expected results in this project. I also thank Rwitajit Majumdar & Shitanshu Mishra greatly for mentoring me in right direction. I would like to thank Prof. Sameer S. Sahasrabudhe and Nitin Ayer for their help.

Contents

List of Figures

Chapter 1

Introduction

A tutor is an instructor who gives private lessons [14]. A good tutor is always needed to learn a subject. A tutor can be a human being or something presented in virtual reality. ***Game based Carrom Tutor*** is an initiative to teach various Carrom related skills through virtual reality. In current scenario there is no such system available in the web for Carrom. This project will endeavour to improve one's Carrom related skills which generally cannot be learnt without some expert assistance. *Game based Carrom Tutor* offers a web based interactive tutorial environment ***Carrom Tutor 1.0*** and a game, namely ***Carrom Tutor 2.0***, where one can learn sophisticated carrom skills.

Now let's concentrate our focus on why *Game based Carrom Tutor* will help in improvement of one's Carrom related skills. Often learning by experience is more beneficial than doing the same by studying. According to the model of *Game Based Learning* the instructional contents are interleaved with the game features (Maja et al.,[11]). The influencing nature of the game compels user to repeat the exercises or quests given in the game over and over until desired level of satisfaction is attained. *Game Based Carrom Tutor* aims at proper mixing of learning content and the game like environment. In *Game based Carrom Tutor* user will be provided with different game like situations where each action chosen by user will lead to different outcomes. This can be directly mapped with Learning Outcome of *Game Based Learning*(Maja et al.,[11]).

Carrom Tutor 1.0 is the first outcome of this project. It is basically a web based system. User can interact with it to watch demos, play exercises etc. It offers the feel of a tutor which provides the knowledge about various carrom skills and asks about their applications in different board positions. *Carrom Tutor 1.0* also provides assessment

and tutoring. Main structure of *Carrom Tutor 1.0* was built using *HTML, CSS, Java Scripts*. *HTML, CSS* were used to build the user interface whereas *Java Script* was used to implement system functionalities.

Carrom Tutor 1.0 lacks in providing flexibility and control to user. To overcome these shortcomings *Carrom Tutor 2.0* is built. To add more flexibility and control it is needed to build a game-like environment. *Blender Game Engine*[7] has been used to create this environment. User can watch demos of different skills and practice those skills right after the demos in *Carrom Tutor 2.0*. A set of complex exercises are built to test user's comprehensibility about the given situations. There are many layers in *Blender* to work with. Each layer has different functionalities. In *Blender*, game objects can be created and manipulated by *Modelling*. Functionalities of game objects can be controlled using *Logic Editor* and *Text Editor* layers. *Logic Editor* offers a large set of logical components to enhance an object's functionalities, whereas *Text Editor* helps manipulating an object's behavior using *Python scripts*.

This project is aimed at the expectation of filling the absence of a good Carrom Teaching Tutor in the midst of numerously available Carrom-Games in the web. It explores two different ways of building the tutor. Learning outcome, interactivity, usability etc. were also compared among these two systems to identify the major factors in building an interactive tutor.

Scope of this project is very large. This project has been initiated as a joint project. Some other related informations can be found in dissertation written by Mayur Katke[9]. Some images used in 3.3, 7 are common in both dissertations.

Chapter 2

Background and Related Work

Teaching properly to facilitate learning, possesses great importance in education. It has been seen over the time that people learn faster while practicing or doing exercises than studying about that matter. But why is it so? This question can be answered in context of ***Game-Based Learning*** & ***Educational Game Design***. Now what is a video game? A video game is one in which a virtual scenario (e.g., virtual world, some simulation, puzzle etc) is presented to the player. The player of the game has to reflect his/her thinking into that scenario by choosing some actions provided in the Game to get desired result. If the expectation is not met the player tries again. But the Game should be interesting enough so that the player will get encouragement to try it again and again until desired result is achieved. If this type of approach is used for educational purpose then it will help in learning also. So in properly built educational game, learner will get involved into the given scenario to find a solution or to achieve the goal. While searching for a solution learner will also need to be able to relate her action and the outcome of those actions. In this way the skills of the learner will improve. Some examples of game-based tutors are given below.

2.1 Examples of Game-based Tutors

2.1.1 System 1

Paul J. Diefenbach[6] mentioned about commercial game development at Drexel University. The author said that while building a game some aspects like fun, strategies, actual

gameplay, the way in which the story of the game is to be presented to the player etc. are generally overlooked. These are very essential for the making of the game. One should not get confused by considering list of game features as the fun part of the game. Instead the fun parts of the game are the aspects of the game through which one can fulfill the sole purpose of enjoyment. Similarly some times the beginners make games with an abstract idea, they may have total knowledge about the story line of the game, even they know the modules to be built, but often it is the case that the actual gameplay i.e. the way in which the game is to be presented to the player is often neglected which causes problems. Third, one should firstly choose how to convey the game to the user. Some common way like animation, comic-like representation, machinima, text format etc have been mentioned in the article.

2.1.2 System 2

System mentioned in Pablo et al.[12] had some interesting features built in it. The most important one is how assessment rules should be applied. User should be assessed with respect to the choices made by her. Suppose there exist many path from start point to end point. Different path provides different situation/environment to users. Activities of user should be checked each time. Different possible activities may differ in their desirability in some particular scenario. So an action made by user in a situation should be rewarded with respect to the most desired activity in that scenario. Negative assessment was also discussed. Negative assessment can restrict a user from doing random activities again and again.

2.1.3 System 3

Features of two tutor generators have been mentioned by Viswanathan et. al.[16]. First one is model-tracing tutors. It focuses upon the underlying process through which user arrives to a state. Second approach, namely constraint based tutoring, does not focus into the fact that how user has arrived to a state, instead it focuses upon whether the constraints by which the states are defined have been satisfied or not. The Constraint-Based Model Tutor(CBMT) is built using JavaCC(Java Compiler Compiler). Relational database had also been used to implement the Constraint-Based Model Tutor and it does not depend upon any programming language. This tutor provides each user same set of problems but with different sets of data. In Model-tracing Tutors(MT) a knowledge-base

has been used instead of constraint-base. The knowledge base is generally a collection of rules. In the implemented tutor Java Expert System Shell(JESS) has been used. In the tutor if a user gives informations about some later problem states then the system detects that some earlier goals have not been satisfied so the current evaluation will be canceled giving the notification of something went wrong.

2.2 Principles for Game Based Carrom Tutor Design

What are the ideal characteristics of an educational game? Maja et al.[11] mentioned that an educational game should be a fusion of pedagogical aspects and game. Also the game should be motivating enough such that a learner keeps on trying over and over to complete the required objective to make an advancement into the game scenario. Inherent property of a learner is to relate the situation presented in Game with real world scenario. If this property of a learner can be exploited in context of Game-Based learning then it can be a great asset to education. Interactivity is another important feature of Game-Based learning. ***Kirkpatric*** mentioned that 'Positive reactions may not ensure learning, but negative reaction almost certainly reduces the possibility of its occuring'(Harold et al.,[8]). Curiosity should also be present in the Game. Curiosity can be maintained by periodic inclusion of new informations into the scheme else the learning will become monotonous. The problems or situations given in the Game should be challenging also. The difficulty of these tasks should be increasing with the increase in levels.

According to Maja et al.[11] & Pablo et al.[12], first step to build an educational game is identifying pedagogical aspects. In other words it can be said that this first step is the answer to the question "What do we want to teach?" In this ***Game-Based Carrom Tutor*** the main aim is to teach sophisticated Carrom-skills to improve learner's carrom playing techniques. Next question is "Why do we want to teach this subject?" The answer to this question is to fill the absence of a good Carrom Teaching Tutor in the midst of numerously available Carrom-Games in the web. Practical implementation of advanced skills and some of the intermediate skills are generally learned by watching an expert playing live. But witnessing such events are quite uncommon in real life. The main aim of this project is to teach various carrom skills through animations and textual descriptions and presenting some board situations to the learner such that the learner can relate some skills and can apply them to achieve the given objectives. This approach is somewhat similar to the ***constructivist*** approach, which tells about learning by making sense of presented material by attending to relevant informations, reorganizing it and connecting

it to what one already knows.

Next step is presenting the scenario to the learner. Main objective of *Game-Based Carrom Tutor* is to build application to teach various strategic and playing skills through demonstrations, exercises, to evaluate users responses to scenarios and to provide allusions whenever needed. Learner will be provided an interface where she can see different skills related to carrom and then the learner's knowledge will be tested against practice exercises with varying difficulties.

Adding assessment is an important step in building educational game. Every situation given to learner should have multiple ways of solving it to achieve the desired goal. Assessment involves analysis of learner's activities which results in change of the given scenario. Learners activities should be evaluated against the best possible solution(s) for each scheme. Assessment procedure should limit learners from trying exhaustively. To reflect this property into assessment deductions should be there for wrong and random choices. Designer should create the assessment rules applicable for the system.

Providing tutorial support to learner enhances the scope of learning. Tutorials are given to a learner to overcome the challenges offered by the problem which the learner could not have solved without assistance. Tutorials are also useful when learners get stuck while trying to progress in wrong direction. Sometimes tutorial support should be given to the user as feedback of the action taken by her. Tutorial support is a great way to increase understandability which leads to learning.

All of these above mentioned features of *Game-Based Learning* and *Educational game* have been considered for designing this *Game-Based Carrom Tutor*.

Chapter 3

Carrom Tutor 1.0

Carrom Tutor 1.0 is basically a web application to teach various carrom skills and their applications to user.

3.1 Design of the Tutor

Design is one of the foremost concerns for building a system. Design of a system is done keeping several factors in mind. As an example different perspectives like Educational perspective, software designing perspective etc. were considered during *Carrom Tutor 1.0*'s designing.

3.1.1 Educational technology perspective

'How people learn' is the main concern of educational technology. Some educational technology principles were adhered to build *Carrom Tutor 1.0*. Educational technology principles which have only been applied in *Carrom Tutor 1.0* are described below. Principles which are common in *Carrom Tutor 1.0* and *Carrom Tutor 2.0* have been discussed in 5.

- ***Recall level questions and Understand and apply level exercises according to Bloom's taxonomy :*** Bloom's Taxonomy classifies human thinking in six cognitive levels of complexity. Revised Bloom's taxonomy has following six cognitive levels.

1. *Recall*
2. *Understand*
3. *Apply*
4. *Analyze*
5. *Evaluate*
6. *Create*

Two types of exercises are given in *Carrom Tutor 1.0*, one is recall level exercise and the other is practice exercise. In recall level one recognizes and recalls facts which she has gone through before. In these exercises an animation of a particular skill is shown to the user and she has to identify which skill is it. There are multiple options provided to users and they just have to click on the button to give their answer. Practice exercises present a set of board situations to learners in which they have to apply skills and strategies which they have learnt previously to finish the game and achieve more points. These exercises cover understand and apply level principles of Bloom's taxonomy. In understand level of complexity, learner grasps the meaning and the concepts behind any task. Afterwards the learner should be able to explain and interpret those learnt concepts. Knowledge obtained in recall and understand level is used/applied in new apply level situations. Every exercise offers different board situation to the learners. Learner needs to apply her skills and suitable strategies to play in it.

- ***Scaffolding***[5] ***:*** Scaffolding is a teaching strategy in which the instructor helps the learner while the learner executes some task. While doing this task the learner faces some difficulties. Instructor's role is to support the learner to overcome these obstacles. Scaffolding helps a learner to have concrete ideas about the actual problems faced during performing a task and their solutions. More about scaffolding and how it is applied in *Carrom Tutor 1.0* has been described in dissertation written by Mayur Katke[9].

3.1.2 Design Perspective

At the very beginning the decision was to provide related exercises after each demo to teach the learner about the use of that particular carrom skill. But in this approach a learner can always try to find the scope to play the particular technique learnt from the

respective demo. In this way the thinking process of a learner will become constrained to some selected techniques. For this reason it was decided later to show all demos to the user in one place and then provide exercises in another place as general exercises which will test the skills of a learner, gathered from all the demos and the prior knowledge she has.

Options selected for *Carrom Tutor 1.0*'s design and reasons behind choosing them

At the index page of *Carrom Tutor 1.0* user will be given the options to view demo, play exercises etc. In exercises, user has options for selecting a striker's position, coin and the portion of the coin to be hit by striker. After selecting these three options player can see the animation of that shot in the given space. While playing a shot on real carrom board, player always positions the striker correctly first and once it is placed player aims at the portion of coin where she should hit to pocket it. Therefore instead of showing dotted line of striker's path, selecting the portion of coin to hit was considered as better option as it is very close to the carrom playing style of any player. Implementing the selection of coin's sector would have been very difficult if player is given the freedom to choose sector according to her wish. To avoid this a large circular image with twelve sectors, designated by twelve buttons as numbers for representing a single portion of coin, is presented in the interface. Player selects the sector of coin to hit by clicking the assigned button(number) on this large circular image.

3.1.3 Architectural view of *Carrom Tutor 1.0*

At the beginning of *Carrom Tutor 1.0* user can select to go for either demos or exercises. In demo page a list of available demos, along with their categorization, will be displayed to user. After user clicks on any one of those demos, corresponding demo of that carrom skill will be displayed to user along with the skill's description. In exercise page user will be shown a list of practice exercises. As user selects any of these exercises, corresponding page for that exercise will get loaded. While playing an exercise user has to select striker position, sector of the coin she wants to hit from a given set of options. If the selection is feasible in terms of pocketing a black coin then the animation of that shot is shown to user, otherwise some messages is shown about the error in selection. In practice exercises user has to pocket all black coins in a row without giving opponent a chance to play.

Activity diagram of *Carrom Tutor 1.0* has been given in dissertation written by Mayur Katke[9]. *Game based learning* principles, described in chapter 2, were also incorporated to enhance learning.

3.2 Implementation

Implementation is one of the integral part of a project. Implementation of the project started after the overall design of the project had been finalized. While implementing this project, preliminary emphasis was to find the suitable platform & tools to build this *Tutor*. Various technological platforms as well as tools have been tried to find the best possible suitability for implementation. Some of these tools & platforms have been taken into account for the actual implementation and rest of them are discarded due to mismatch with the scope of this project. Some selection guidelines have been mentioned by Paul J. Diefenbach[6]. Some tools like *Unity, OpenGL, Java Server Pages* etc. were considered for developing, but were rejected due to mismatch with various development constraints. *Carrom Tutor 1.0* was built using *Html*, *CSS*, *Macromedia Flash MX - trial version* & *Java Script*. *Html* and *CSS* are being used for creating the web pages and making them attractive to user. While *Java Script* is responsible for implementation of internal logic related to user response and corresponding action taken etc.

Macromedia Flash MX - Trial version

This tool is mainly used for creating *flash* animations. While making the animations many layers were created for each object. A time line is given in this tool to manipulate objects efficiently. Producing animations using this tool is relatively easy, but the process of arranging the layers, timing the activities in those layers are some what tricky. All animations used in *Carrom Tutor 1.0* are created using this tool.

HTML

HyperText Markup Language(HTML) is a type of markup language which is used to create web pages and provides many options which are called *Html elements* to manipulate the structure of the web pages according to one's need.

The *Demo.html* page offers the scope of watching different *Carrom skills* and their

descriptions to user. In this page at the left side different *Demo skills* along with their classifications according to execution difficulties have been presented. At the right hand side of the page *Demo shots* are displayed by showing corresponding animated *.gif* files. In the middle there are some text description about each *Demo shots*. In the *Demo* page at the top portion there is a *div* which has the required links to *Home* and *Exercise*. The next *div* is the main portion of the body of this page. This *div* has been subdivided into three different *div*s. These three *div*s occupy $20\%, 25\%$ and 55% of the total horizontal width of the screen respectively from left to right. In the subsequent part of this report these three horizontal *div*s will be referred as left, middle & right for ease of understandability. This distribution has been followed for rest of the pages to standardize the overall appearance of pages.

In *Recall.html* the left *div* shows exercise names as list and in the middle multiple choices have been given to the user, as possible answers to the question, but only one of them is correct. In the right *div* animations are displayed. Links to other pages have been mentioned in the *div*.

More description about *HTML* implementation can be found in dissertation written by Mayur Katke[9].

CSS

CSS is responsible for the designing of the pages and their appearances to user. A file named *mystyle.css* has been created to store all *CSS* properties used for this project and it has been included in all *.html* pages to reflect those modified properties.

In *Demo.html, Exercise_Page.html, Exercise_i.html* and *Recall.html* the options (it can be for exercise or demo shots or input selection for exercises etc.) are displayed as list with some particular appearance properties. General list properties used in *mystyle.css* for this project is given on the next page.

Afterwards each list item is considered as a *html - button*. Then in *mystyle.css* different classes of different *button*s have been created with different properties to distinguish between their actions. Later appropriate *Java Script* functions have been invoked using *onclick* property of *button*s. Example related to *button*s can be seen in dissertation written by Mayur Katke[9].

```
#navcontainer ul
{
   margin: 0;
   padding: 0;
   list-style-type: none;
}
#navcontainer a
{
   display: block;
   color: #F3C;
   background-color: #036;
   width: 11.1em;
   padding: 3px 12px 3px 8px;
   text-decoration: none;
   border-bottom: 1px solid #006;
   font-weight: bold;
}
#navcontainer a:hover
{
   background-color:#060;
   color:#FFF;
   transform: scale(1.05) translateZ(0);
   text-shadow: 0 1px 1px rgba(0,0,0,.3);
   -webkit-border-radius: .5em;
   -moz-border-radius: .5em;
   border-radius: .5em;
   -webkit-box-shadow: 0 1px 2px rgba(0,0,0,.2);
   -moz-box-shadow: 0 1px 2px rgba(0,0,0,.2);
   box-shadow: 0 1px 2px rgba(0,0,0,.2);
}
#navcontainer li li a
{
   display: block;
   color: #FFF;
   background-color: #0C3;
   width: 9em;
   padding: 3px 3px 3px 17px;
   text-decoration: none;
   border-bottom: 1px solid #0F6;
   font-weight: normal;
}
```

Java Script

Java Script handles the main part of the system implementation. It interacts with the user inputs and changes the web page contents accordingly. *Java Script* code for a web page is embedded at the very beginning of it's *html* page. For *Demo.html* a *function change(num)* has been created which takes an input value according to the user's chosen demo shot. This input value is used to find a match in a list of *switch cases*. After a

match is found, corresponding functionalities change some specific part of the *html* using ***innerHTML*** property. This *innerHTML* is known as *HTML DOM* which provides the facility for accessing and manipulating *HTML* documents. The right *div*(where the animations of shots are shown) has been assigned *id* = *"content"* and the middle *div* has been assigned *id* = *"skill"*. Function *change()* changes the content of these *div*s to appropriate content (i.e., animation and text) depending upon the input taken.

In *Exercise_i.html*, the main functionality of displaying the proper animation according to the user inputs is managed by a function named *Play()*. Some variables, namely coin, sector, striker, have been used to store the current choices of user. While user clicks on a particular option for coin, sector & striker, *setcoin()*, *setsector()* & *setstriker()* functions are called respectively to set the current value of these three parameters. While user clicks the *Play* button it invokes *Play()* function to show the corresponding animation or feedback. Instances of *setcoin()* & *setstriker()* from *Exercise_2.html* are given on the next page.

```
<ul>
  <li> <p id="listHead">Choose Coin</p>
    <ul>
      <li> <button class="button" id="save"
        onclick= "setcoin(1);setvalCoin('Coin 1')">
        Coin 1</button> </li>
      <li> . . . </li>
      <li> . . . </li>
    </ul>
  </li>
  <br>
  <li><p id="listHead">Choose Striker Position</p>
    <ul>
      <li> <button class="button" id="save"
            onclick="setstriker(1);setvalStriker('Striker 1')">
            Striker 1</button> </li>
      <li> . . . </li>
      <li> . . . </li>
    </ul>
  </li>
</ul>
```

In these pages the *id* of right *div* is "gif". Clicking of *Play* button takes the current values stored in different variables and starts executing the *Play()* function. Some part of the *Play()* function is given on next page. This function checks values of different variables and flags, then according to those values it examines different conditions given

in the program to display appropriate animation or message. In this function *innerHTML* is used to change the animation of the rightmost *div* and *setTimeout()* function has been used to provide time delay wherever necessary. Frequent usage of *alert* boxes has been made to display the messages.

```
function Play()
{
   Gif=document.getElementById("gif"),
   GifContent = Gif.innerHTML;
   if(coin==2 && striker==1 && sector!=3 && flag==0)
   {
      if(sector==2 || sector==4){
    choice=100;
      }
   else{
      choice=2;
      }
   }
   if(coin==2 && striker==1 && sector==3 && flag==0)
   {
    choice=6;
   }
    .
    .
}
```

Here choice 2, choice 100 lead to two different error messages with some point deductions. On the other hand choice 6 changes the contents of rightmost *div* using *innerHTML* and displays convenient animations.

3.3 Demonstration

This chapter gives a rough sketch of *Carrom Tutor 1.0*. Index page displays five blocks with different names. Once user uses mouse to hover upon them, some text description telling about the possible contents of them shows up. User can explore different pages containing demo, exercises, related documents, game rules and about the developers.

Demo page is shown in the figure 3.2. List of demos for different skills are listed at the leftmost div. These skills are also divided into three categories basic, intermediate and

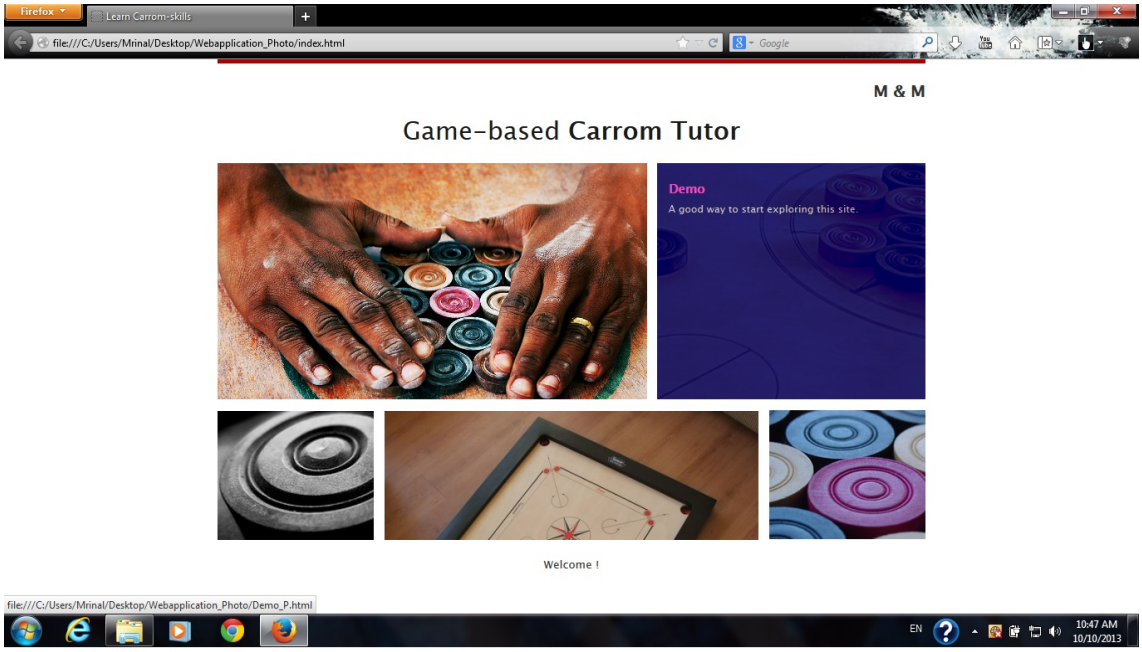

Figure 3.1: Index page

advanced. If user clicks upon any of these buttons then the corresponding demo will be shown to the right most div, which shows the picture of a carrom board. A few words about those skills also appear whenever the corresponding button is pressed.

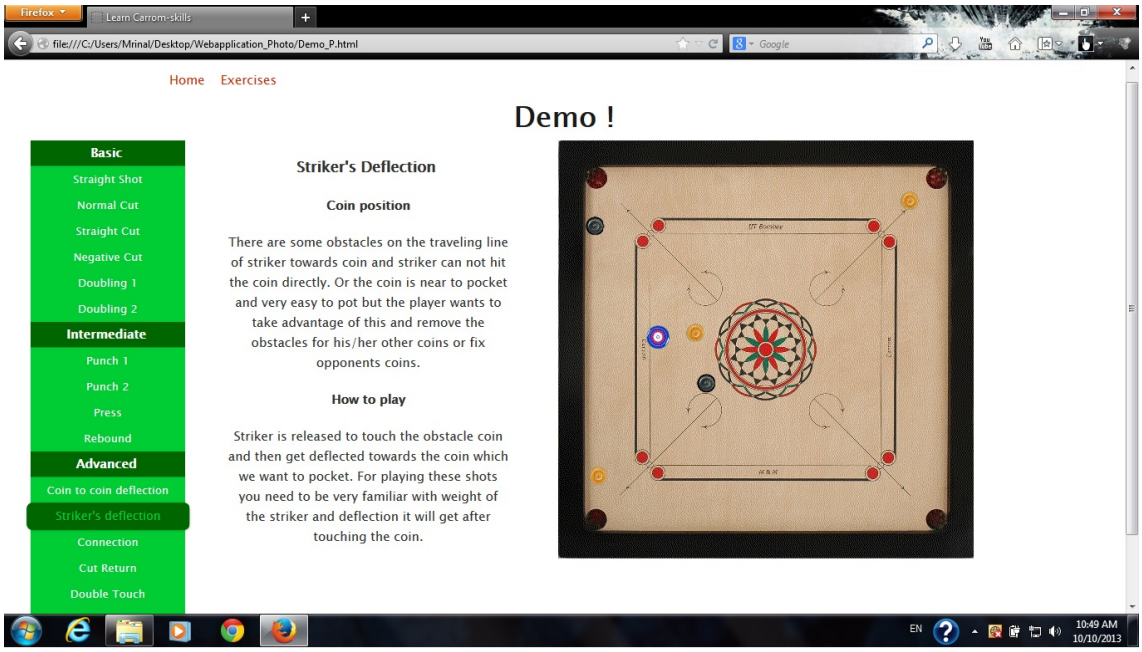

Figure 3.2: Demo page

Exercise_Page.html (Figure 3.3) displays list of practice exercises at the left. At the

right most div practice exercises get loaded. In the middle portion of the page some general instructions have been given so that user does not face difficulties while playing the exercises.

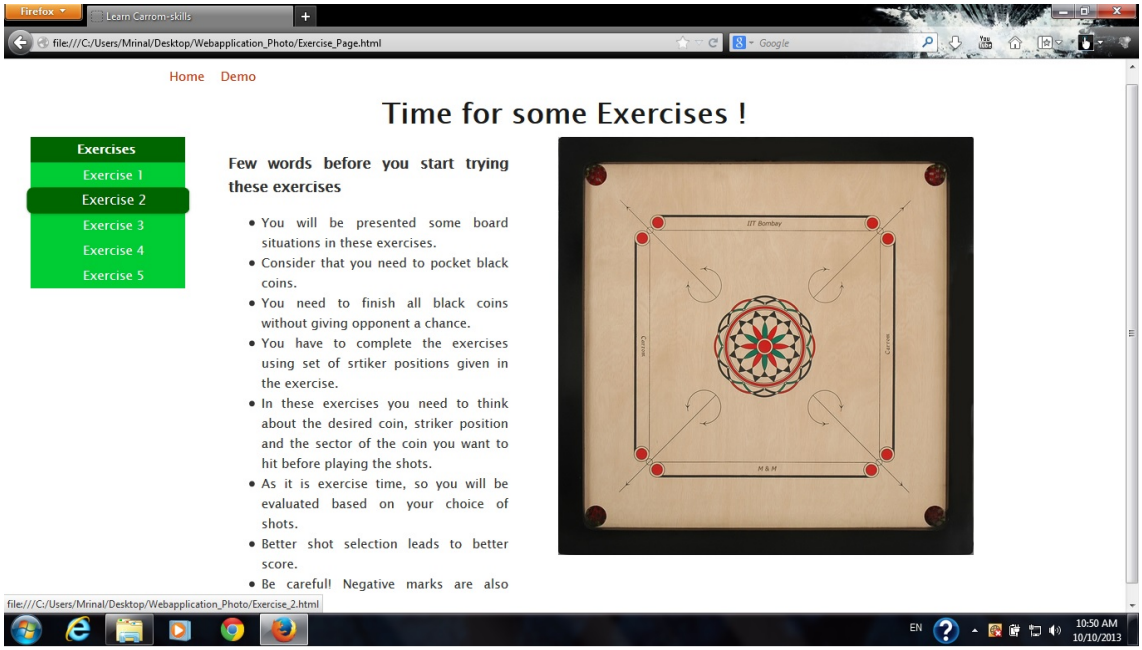

Figure 3.3: Exercise page

Figure 3.4 shows the general structure for the page of recall level exercises. At the left side of this page exercises are presented in listed form. In middle *div* choices for each exercises have been presented. If any recall exercise button is clicked, a demo will be shown to user in right most div. Middle section of this page contains a set of possible answers. If user clicks the right option then she will be congratulated by system else will be asked to try again.

Figure 3.5 shows the structure of *Exerise_i*th page. Left div displays options for striker position and the coin to hit. Middle div shows a circular image which represents coin. It is divided into 12 sectors, each having a clickable button. Below this image there is a speech box which stores options chosen by user. Rightmost div displays the animation of some specific shots.

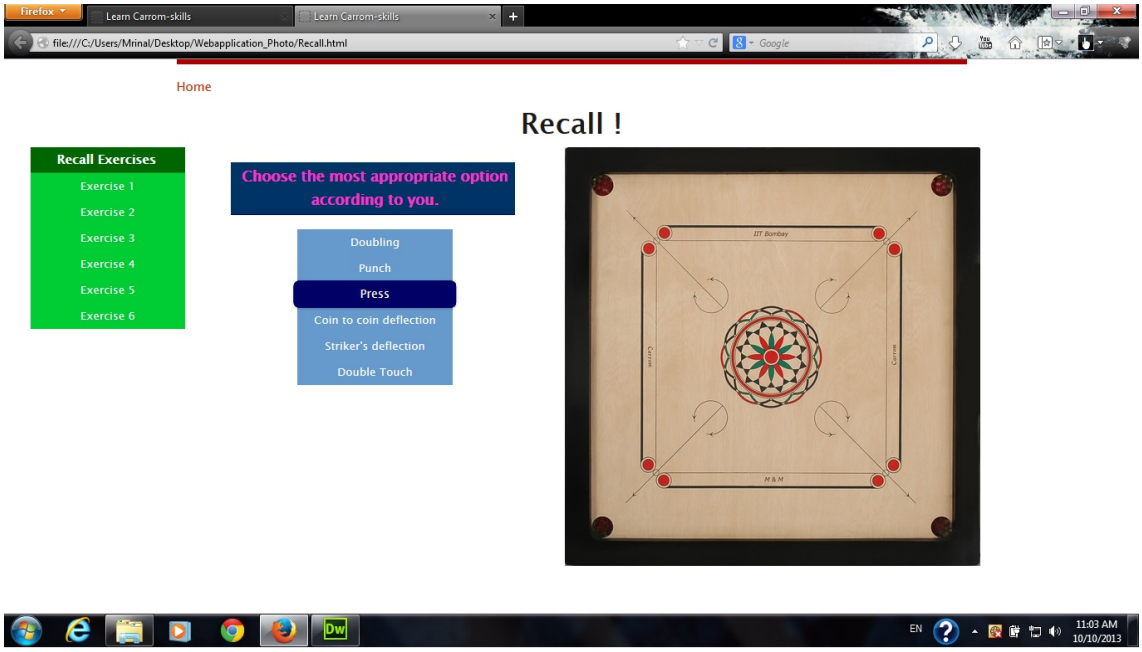

Figure 3.4: Recall exercise page

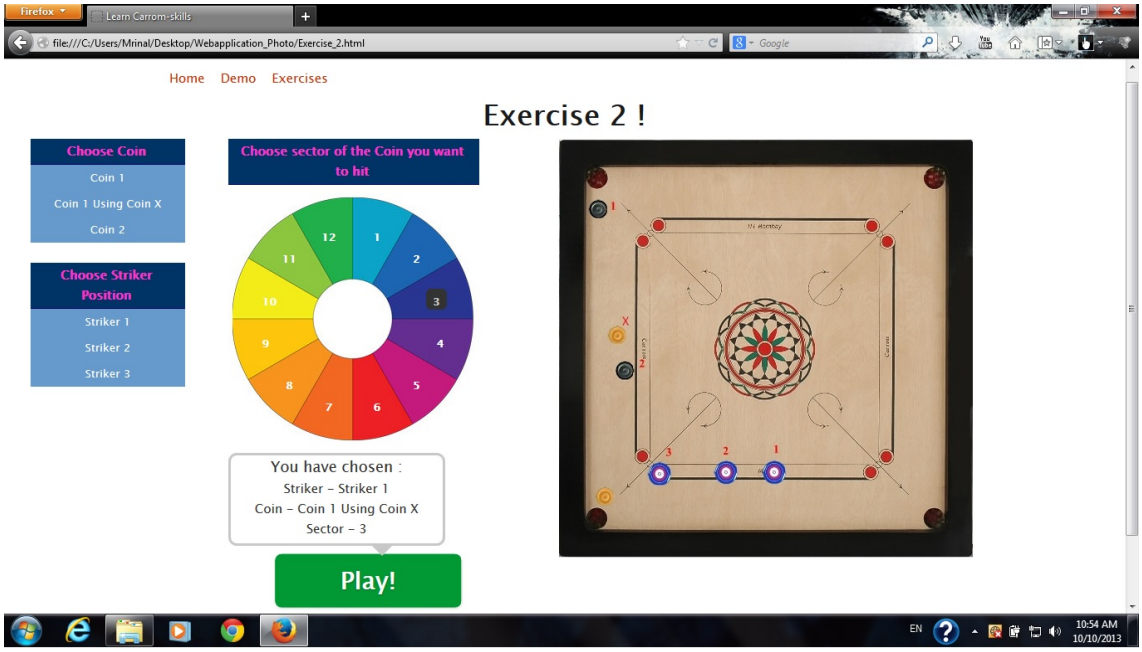

Figure 3.5: Exercise 2

3.4 User Experiment

Description of user experiment done for *Carrom Tutor 1.0* has been described in this section. Firstly the sample ,i.e. desired audience, then data collection procedure has been mentioned. Later analysis of the collected data has been given.

3.4.1 Sample

Any one who has a basic knowledge about carrom and played carrom game few times can participate in the user experiment. Seven users participated in *Carrom Tutor 1.0*'s user experiment. They were basically beginners and intermediate in carrom.

3.4.2 Data Collection Methodology

The experiment has been conducted in two phases. In first phase user was asked about her interest, expertise in carrom. Some complex board situations were given to her and asked which shots she wants to play. In second phase user is allowed to interact with the system in a step by step manner. Again those board situations were presented to her to check the change in her thinking process about that board situation. More details about data collection methodology can be found in dissertation written by Mayur Katke([9]).

3.4.3 Data Analysis and Results

SUS analysis[2, 4] have been used to check the usability of *Carrom Tutor 1.0*. *SUS* analysis asks 10 questions to check any system's usability,attractiveness etc. These ten questions and their responses from users are summarized below. The first question asks about the willingness of user to use this system in future. The surprise was that all users "agreed" to use this system in future. This result is quite motivating for the initiative. All most all of them think that this system will help to increase their knowledge in Carrom. Question 2 & 3 enquire about the simplicity of ***Game-based Carrom Tutor***. Most of the users "disagreed" with the statement that the system was complex to use. Remarks of the type "Too simple and lucid" were given by some of them. Question 4 asked whether the user needs the help of a technical person to use the system or not. Some users "strongly disagreed" while the rest "disagreed" about the necessity of technical assistance. Next two questions were about how well the system is integrated and whether too much inconsistency is present in the system or not. Responses given to us were mixture of "neutral"s and "agree"s for the integration aspect. All most all of the users "disagreed" with the statement about the presence of too much inconsistency. Most of the users "strongly agreed" or "agreed" with the statement of learning the system quickly. Though some responses were given saying that it might take some time for the beginners to get used to with the system. When asked about whether the system is cumbersome to use or

not the answer were of the type "strongly disagree" or "disagree". Some suggestions for proper arrangement of the instruction were given. Whether a user was confident enough or not while using the system is asked in question 9. Given answers range from "neutral" to "strongly agree". Some confusion about the presentation of the exercises were noted. Last question asks if one needs a lot of prior knowledge about the domain to interact with the system or not. All most all of the users "strongly disagreed" or "disagreed" on the necessity of having vast prior knowledge. Average *SUS* score for *Carrom Tutor 1.0* is 77.14 out of 100. A *SUS* score of 68 represents average usability. So *Carrom Tutor 1.0* has a significantly higher score in terms of it's usability. Each *SUS* analysis question and their average responses from users have been shown in figure 3.6. Integers ranging from 1 to 5 denotes Strongly Disagree, Disagree, Neutral, Agree and Strongly Agree respectively. For an example let's consider the first column of figure 3.6. This column shows the average response given by users for the first question which asks whether the user wants to use this system frequently in future. Score of 4 means "agree". So the users agreed to use this system in future frequently. Similarly other questions' average score can be seen from the respective columns.

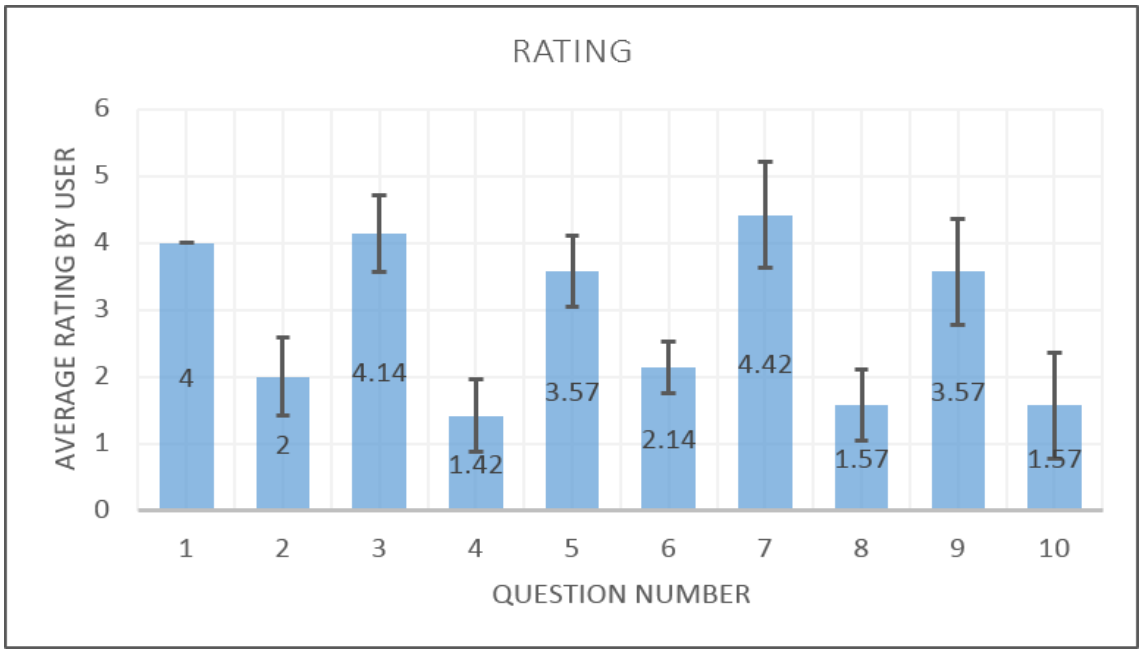

Figure 3.6: SUS feedback

Chapter 4

Motivation for Carrom Tutor 2.0

Initially the tools and technological platforms for implementing *Carrom Tutor 1.0* were not decided. Some initial experimentations were done over several possible choices. Game engine ***Unity*** and 3D graphics software ***Blender*** were studied for checking the suitability with required design of the tutor. The plan of using any of the two game engines for implementation was dropped because game engines are very vast and learning to use their functionalities takes more time. Also the design was very simple as compared to game engines' functionalities. Secondly for creating graphical representation of carrom board and coins some basic programming in ***OpenGL*** was done. But implementing whole system in OpenGL is very difficult, as controlling the movements of objects following the laws of physics is very challenging to implement in OpenGL and it might deviate from the scope of the project. Later, to create the animation of shots ***Macromedia Flash MX*** was chosen because with some set of images on timeline, animations can be generated in this tool easily. Animations of all shots and exercises used in *Carrom Tutor 1.0* are produced using Macromedia Flash MX.

Choice of language for programming or web application designing is one of the main steps in implementation. Suitable language should be chosen on the basis of sufficiency provided to the objective by its functionalities and on the basis of developers'comfort level while working with it. Programming languages like ***Java, PHP, C++*** were contemplated for building a good interface for tutor with animations. But as discussed in 3.2, implementation of *Carrom Tutor 1.0* was started in ***JSP***. Displaying the animations of shots and taking inputs from user were implemented into the system, but there were some problems while changing the contents of web pages dynamically in JSP. So, the first

exercise was implemented completely using the brute-force approach in which a separate web page was created for each shot played in the exercise. Most of the designing part of code was redundant in this exercise and for each button you click or any action you do separate web page was getting loaded in browser.

Carrom Tutor 1.0 offers a set of pre-decided positions for striker to play with. Moreover, all possible shots from those pre-decided positions were not shown to user. If user selects a shot, may be or may not be optimal shot, which pockets user's coin then only that shot is displayed to user using animated *.gif*s. So in *Carrom Tutor 1.0* certainly there exists an issue with flexibility given to user. Apart from user flexibility, the way in which exercises were presented to user needed to be improved. In other words *Carrom Tutor 1.0* delivered the knowledge in a crude manner. There is a need to enhance the tutoring procedure by keeping the original flow used in *Carrom Tutor 1.0* intact.

4.1 Design Decisions for Carrom Tutor 2.0

Blender Game Engine[7] has been used to create second version of the tutor, namely ***Carrom Tutor 2.0***. It is actually a game which teaches carrom skills to user. User can play any shot from all possible and permissible striker positions. User can also select the desired force to release the striker. In this game user can experience actually what happens inside a real game if a shot is played. There was a problem of creating more and more *.gif*s as the complexity of the exercise goes up. In a game like situation there is no need to create any thing in advance, every thing happens in-game. *Carrom Tutor 1.0* offers a top view of the carrom board while playing the exercises. If the actual user's view of the board during real life carrom game can be presented in the tutor then the experience will become more intense. This ***3D*** view has been incorporated with the new system. *Carrom Tutor 2.0* adjusts the camera view according to the mouse movement of user.

Carrom Tutor 2.0 offers animated videos instead of *gif*s. These videos demonstrate different carrom skills. In *Carrom Tutor 2.0* exercises have been designed inside the game-like environment. Two types of exercises have been designed. First one is practice exercises. Practice exercises generally test user's understandability about the skills shown in demo. Complex exercises test user's carrom skills in relatively complex carrom board

situations. In a complex exercise, shots played by user are assessed against the optimal possible shot(s) that can be played in the given situation. User will be given scores based on the shot(s) played by her. User can browse through different carrom game strategies and game rules in *Carrom Tutor 2.0*.

Chapter 5

Design of Carrom Tutor 2.0

Design of a system involves it's underlying architecture, structural components, user interface etc. Some of the design principles were mentioned in 3.1. Some common design principles were adhered to build these two tutors. Details of these design principles are given below.

5.1 Educational Technology Perspective

Main aim of this *Game-Based Carrom Tutor* is teaching various carrom skills and strategies ranging from basic to advanced to users through an interactive tutoring environment. Once the system is experienced by a learner, the expectation will be that for a given board position she should be able to make decisions about the necessary choices of skills and strategies to be executed in order to get more advantage in game. Design of *Carrom Tutor 2.0* is based on cognitive theory of learning to facilitate the learning of user. While constructing this tutor the main focus was upon human learning process and features to be used to promote learning. At the beginning, the user is shown demos of various possible carrom skills using videos and some text explanations embedded in it. Next practice exercises for those skills shown in demo will be given to user so that user can learn to properly execute those skills. Then the user will be provided with board situations in form of complex exercises in which she has to apply playing as well as strategic skills to pocket all coins without giving a chance to the opponent and while doing this she will earn some points. *Carrom Tutor 2.0* is built by exploiting different features of learning theories and principles. These are given below.

- ***Cognitive Model of the Mind :***

  An abstract representation about the working procedures of human memory is given by Cognitive model. There is sensory memory, working memory (short-term memory) and long-term memory in human brain and the information is processed in them with same order. Working memory is limited in capacity but speed of processing is very fast. On the other hand long-term memory has infinite capacity and permanently stores informations, but information loss may happen if those informations are not used for a long time. Relevant information is processed, organized in a coherent structure in working memory and then transferred to long-term memory.

  Working memory is limited but it has **separate channels** for processing **visual** and **verbal** material. In *Carrom Tutor 2.0*, while teaching a particular skill a video of that skill is shown to user. Text instructions for a skill is provided inside the corresponding video of that skill in order to utilize both visual and verbal channels effectively in learning. This leads to active processing of the information and better learning.

  Active processing of informations becomes difficult when information provided to working memory exceeds the limit. Memorization capacity can be increased by ***Chunking*** i.e. grouping small & related pieces of informations into larger, meaningful units. This task of *Chunking* is simplified in the tutor when user is learning skills by watching videos.

  Skills that are taught to users through demo are also divided in three categories namely basic, intermediate and advanced.

- ***Modelling & Sequencing*** [5] ***:*** Modelling means learning by observing. The basic idea is to show learner how a task is performed by an expert. Later the learner tries the same. Sequencing focuses upon proper arrangement of different aspects of the subject. Proper arrangement of the learning activities helps learner to grasp the ideas easily and making the conceptual model in mind. Details about these two principles can be found in dissertation written by Mayur Katke[9].

5.2 Perspective of Game-Based Learning

Desired features of *Game-based Learning* have been discussed in chapter 2. Firstly to make the game interesting the *User Interface* has to be attractive. Next step is making the game motivating. To make it exciting some reward scheme can be included into the game aspect of the system. There should be a highest score, and the user will be rewarded with some score for each successful action. In the end one can compare her achieved score with the highest possible one. If expectation is not met then she can try again. As it was mentioned in *Design perspective* two different types of exercises have been designed in *Carrom Tutor 2.0*. Practice exercise are relatively smaller exercises in which user has to apply a particular skill shown in corresponding demo. Instead of using point system, feedback texts have been used as reward. If a user executes the desired skill in a practice exercise and manages to pocket her coin then the system displays text to congratulate the user. If the opposite happens then the *Carrom Tutor 2.0* reloads the board situation. Sometimes feedback is given to user as a hint to apply the respective skill if user is unable to perform that specific skill. Now let's talk about the reward system for complex exercises. Evaluation of user selected shots have been incorporated in complex exercises. For each of these complex type exercises highest score is fixed to 100, i.e. a user can score at most 100 points in any complex exercise. Each successful shot of user will reward her some points. Shot played by user has to be compared against the best possible shots that can be played in that situation. If a user plays a shot which pockets desired coin(s) but the shot is non-optimal then the system will reward relatively less points to user. At the end of a complex exercise if it is the case that user successfully finished the exercise but the total points earned is less that 100 then user will understand that somewhere her shot selection(s) was/were non-optimal. So she can try again to score maximum points.

The game should be presented to the user in such a way that she finds it easy to relate with real world scenario. To satisfy this necessity the type of options given to the user to execute a carrom-shot should have similarities with executing the same in real Carrom board. User has been given complete flexibility to play any shot in a given situation. To match the experience of playing a shot in *Carrom Tutor 2.0* and on real carrom board it was needed to incorporate user's view of the carrom board. If the actual view of a carrom board from a user perspective can be offered then one can relate it with the actual carrom game experience. ***3D*** view of the carrom board has been provided for this purpose. In reality when one is playing carrom then she aligns herself at different angles with respect

to the carrom board to play different shots for the ease of executing those shots correctly. In *Carrom Tutor 2.0* user only has to move the mouse to get the desired view of the board from different angles. Secondly, user can place the striker at any permissible position on the baseline. Ever on the fly coin detection has been included in the system. After user plays a shot it should be checked that if any of the coins stops on baseline or not. If it is the case then user should be restricted to place striker on a position which conflicts with that coin's position. Thirdly, the angle of seeing the carrom board is also applied on striker to match the rotation of the striker with mouse movement. Lastly, user can also select the force by which she wants to hit a coin. A power slider is shown on-screen to help user selecting the desired force. This slider goes up an down. When user performs second mouse click then the force proportional to slider's height is applied on the striker.

Proper feedback should be displayed to the user. As mentioned previously in practice exercises feedbacks have been given to either congratulate or to provide support to user. If user successfully finishes a complex exercise with optimal shot selections then the system congratulate her mentioning her optimal shot seletion.

Properties of this system matches the functionalities of ***Model-tracing Tutor*** mentioned by Viswanathan et al. [16]. *Model- tracing Tutor* is a type of *Intelligent Tutoring System*. *Model-tracing Tutor* emphasizes upon the undertaken steps through which a user arrives at a particular situation. *Carom Tutor 2.0* also emphasizes upon the fact that whether user has executed optimal shots or not. Tutorial support should also be given in form of feedback message to the user whenever necessary.

5.3 Design perspective

Two types of exercises, practice exercises and complex exercises, have been designed for better learning. Each tutorial video is followed by practice exercise(s). Complex exercises which is given separately, are relatively large exercises and complex. These exercises generally have more than one possible way to finish. Users will be awarded with marks according to the path taken to finish the exercises. Details about how to play shots are described in dissertation written by Mayur Katke[9].

Initially it was decided that there will a bar having divisions for different force values. User needs to click on the desired value and the respective force will be applied on the striker. But now a days most of the games use a power slider which oscillates between a given highest and lowest value. And it is more attractive. For this reason this oscillating power slider was considered for *Carrom Tutor 1.0*.

To assist user, if the actual direction of striker can be shown then it will be very useful for them. In order to do this a red line is displayed which indicates striker's path after it's release. Now we can extend this concept further. If the path along which the striker will get deflected after first hit is also displayed then also it will become helpful. But this will reduce the complexity of the exercises. One can take a shot by seeing the striker's trajectory only. There will be no need to visualize the shot in mind before taking the shot. In real life carrom game there is no such type of assistance. So it was finalized to restrict the visualization of striker's path up to first target.

5.4 Architectural view of the Tutor

User can navigate to tutorials or can try complex exercises after starting the tutor. In tutorial screen user can watch the demo of her desired skill. After watching the video user will be presented with practice exercise(s) similar to the situation presented in the video of that skill. User can either complete the practice exercises of a skill or navigate back to the tutorial screen. If user can not apply the desired skill in practice exercises then the exercise will get reloaded. In some practice exercises some hints are also given in form of feedback to remind which skill to apply.

In exercise screen a set of complex exercises have been given. These exercises requires two or more shots to be played successfully to get finished. These successful shots need to be played consecutively. In other words user has to pocket all black coins in a row without giving opponent a chance. Failure to do this will result in scene reload. In these exercise users will be given scores according to her performance. Activity diagram of *Carrom Tutor 2.0* is given in figure 5.1.

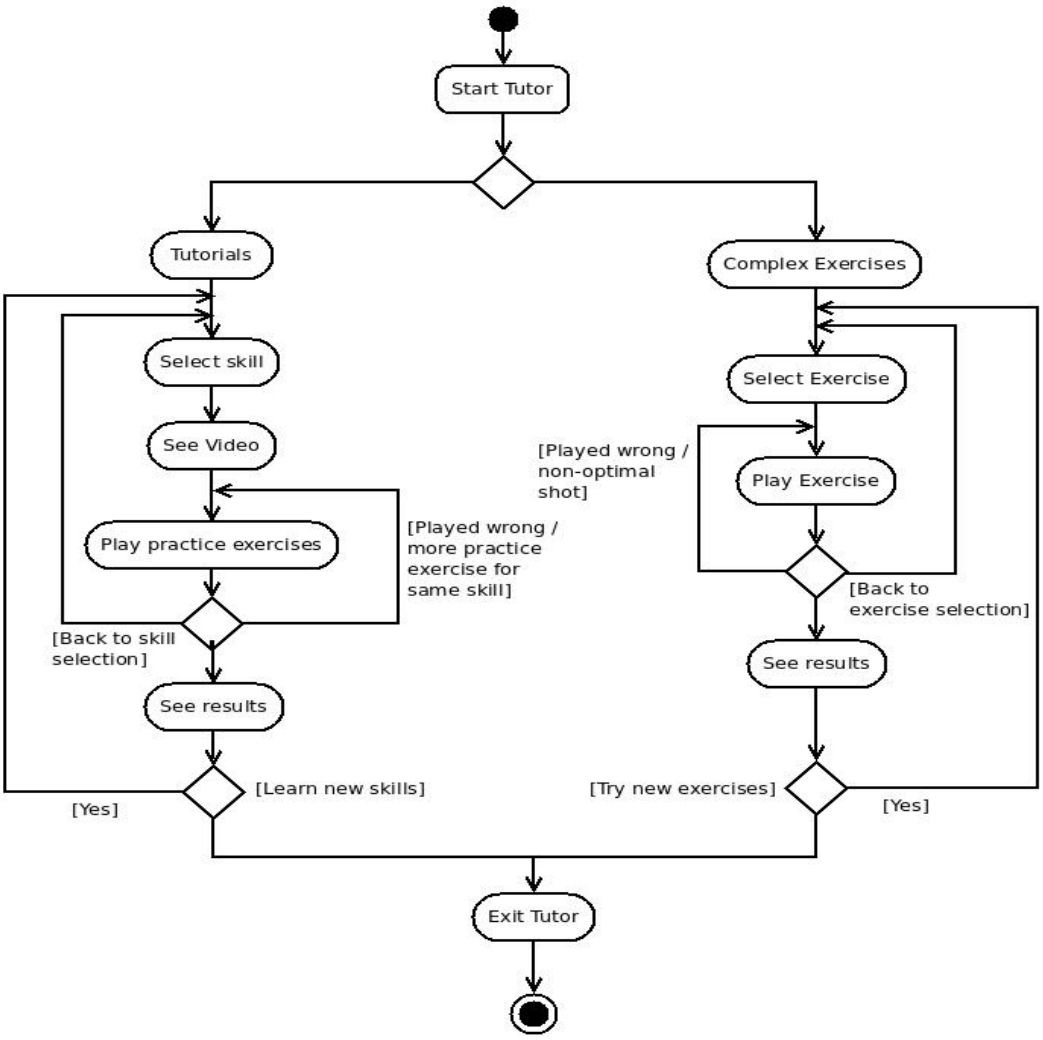

Figure 5.1: Activity Diagram for *Carrom Tutor 2.0*

Chapter 6

Implementation

As mentioned in Chapter 4, there were several concerns about how the system should be presented to the user. One of those main concerns was offering more flexibility to user so that user can play any shot according to her wish. Feel of a real carrom board game should be present. A game-like environment was needed to be built. These requirements can be met if the system is implemented using a *Game Engine*. Now a days games are generally built using *Game Engines*. Game engines provide very interactive environments to create games. Newer game engines allow to add more features inside the game. There are many game engines available on the web. *Blender Game Engine*[7] is one of them and it is an open-source software. *Blender* is also one of the most popular open-source game engines available.

Carrom Tutor 2.0 has been created using *Blender Game Engine*. Implementation using *Blender* has three main parts - object modelling, logic editing and python scripting. Modelling allows to create, transform and manipulate game object properties efficiently. Logic bricks and python scripting are used to implement functionalities in the game. Screen-shot of *Blender*'s user interface has been given in figure 6.1.

6.1 Modelling

Modelling means creating objects, transforming them according to need, setting up their properties etc. *Blender* provides very interactive layers to do modelling. Carrom board is made up with cubic mesh objects. While interacting with the tutor one can see that the

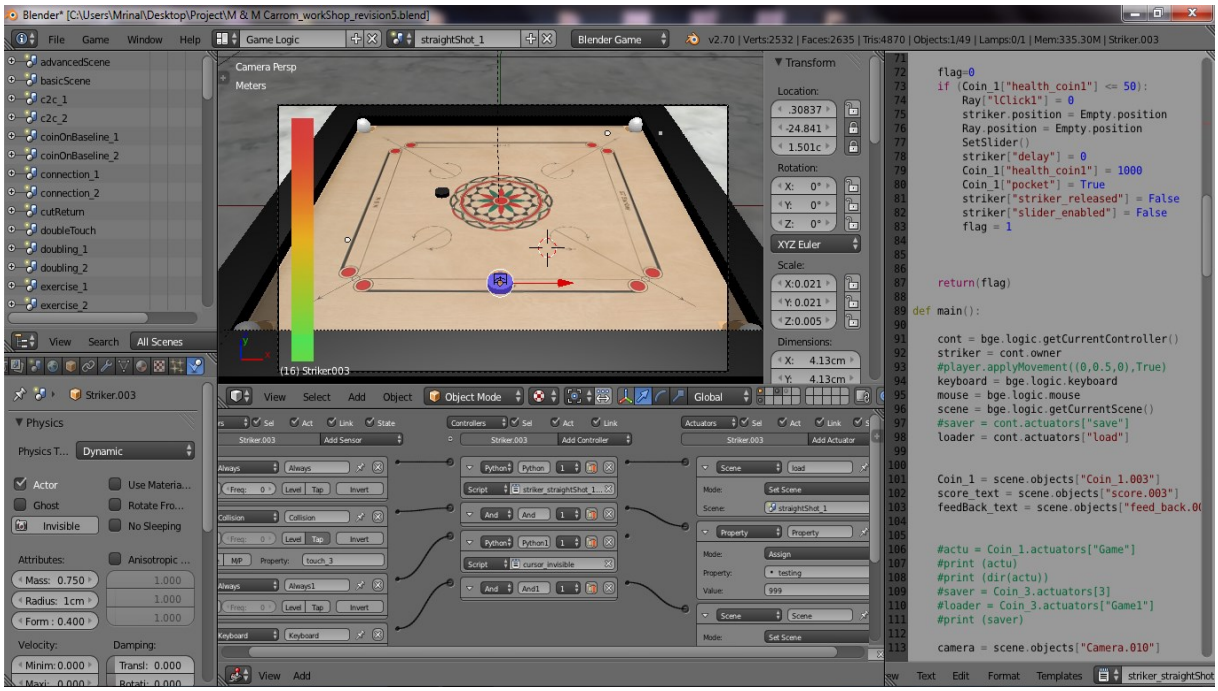

Figure 6.1: User interface of *Blender Game Engine*

carrom board is inside a room. This room has been designed by creating plane objects and unwrapping some texture images onto them. The power slider bar, and other objects which are displayed to the screen, have been attached to the camera object so that they can move along with the camera. More details have been stated in 9.

6.2 Logic Editor

Blender Game Engine has a scripting layer called *Game Logic*. Game logic has three main parts *Logic Bricks*, *Properties* and *States*. Most important part of this *Game Logic* is *Logic Bricks*. *Blender Game Engine* provides a very interactive graphical interface named *Logic Editor*. *Logic Editor* provides rectangular logical blocks to create the desired functionalities inside the game engine. For these rectangular blocks *Logic Editor* is commonly referred as *Logic Bricks*. We can set up a collection of logical components for each game objects. These logical components can be combined with each other to get the desired functionalities out of the system. There are three types of components *sensors*, *controllers* and *actuators*. *Logical connectors* are there to establish connections between these trio. *Sensors* are related with sensing specific situations/properties etc. *Sensors* are triggered when something desirable happens according to the functionalities set into the *sensor(s)*. When a *sensor* is triggered it sends out positive pulses. Moreover these pulses

can be set up as continuous mode or instance mode. *Controllers* are required to get the outputs from *sensors* and fed them either to *actuator(s)* or to a *python* script attached with that corresponding *controller*. *Actuators* are there to perform certain tasks set up by users. A lot of functionalities can be performed by proper use of *Logic Bricks*. Some important modules done in *Logic Editor* are described below.

1. **Designing UI**

   The user interface consists of many scenes created inside the game. Different game objects are needed to create these scenes. Some objects act like buttons in these scenes where users has to click to interact with. More details of these game objects are described in 9. In the start scene, namely startScene, there are sensors attached with each objects created for the user to get interact with. 'start_sensor' is one such object in start scene. Mouse clicks on these clickable objects are captured with two sensors of type 'Mouse ' . One of them sends positive continuous pulse when mouse is hovering over that object using 'Mouse Over' event while the second one gives a positive pulse instance when mouse is clicked. There is another 'Property' *sensor* which triggers a positive pulse when the value of variable named 'animationProp' has False value. These three *sensors* are attached to an 'And' *controller*, that means when these three *sensors* send positive pulses together then only the corresponding *actuators* execute the desired function assigned to it. Two *actuators* are connected with the 'And' controller, one of them is of type 'Action' type and responsible for showing a small animation created with respect to the scaling of the object it is attached with. Second *actuator* is a 'Scene' type and responsible for changing the scene to another scene named *secondScene*. Figure 6.2 given below describes it's *Logic Editor* settings.

   Basic, Intermediate and Advanced tutorials are shown in three different scenes namely basicScene, intermediateScene and advancedScene. Logical components and their interactions in basicScene have been described here as an example. In this scene there are three objects reacting with user to get the choice of tutorial difficulty(basic, intermediate, advanced). If user clicks 'Basic' then this scene displays the basic carrom skill options by changing their 'visibility' property. Let's describe it a little further. Consider the first option under basic tutorial, *Straight shot*. There is a text object displaying 'Straight shot' . Underneath it there is a plane object which is connected with *Logic Brick* sensors. The text object is a child object of this plane. This plane object is named as 'straightShot' and it is set invisible. It is associated

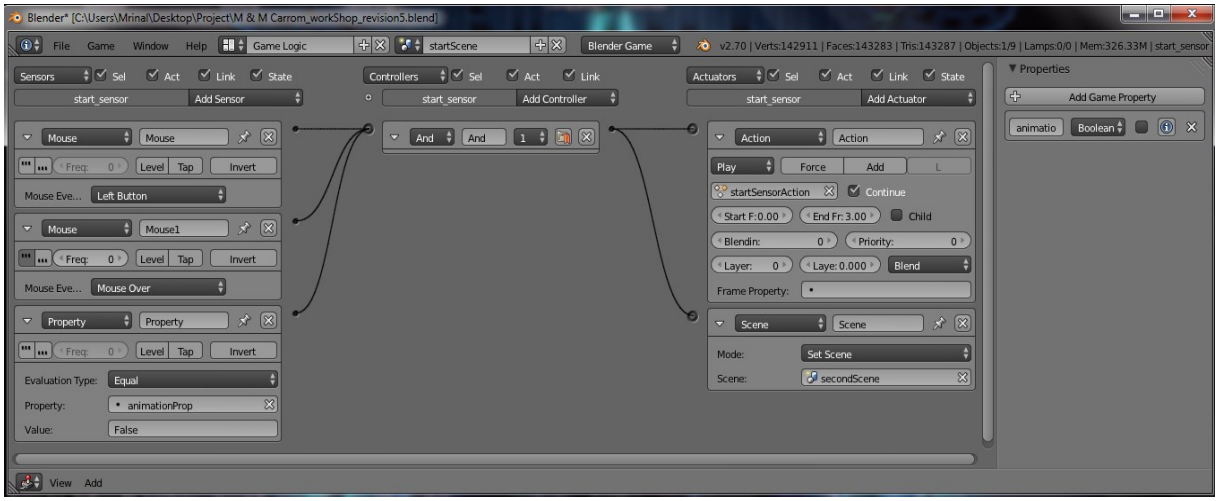

Figure 6.2: Logic bricks for start scene

with two global variables in this scene namely *basicVisi_1* and *animationProp_1*. Both of these variables are set *False* initially. There are two 'Property 'sensors associated with the value stored in *basicVisi_1*. If the value is false(default value) then corresponding 'Visibility' actuator makes it's children, i.e. text object, visible. If *basicVisi_1* is *True* then there are two actuators, one of them makes both of the plane and text visible. Second actuator makes the plane(parent) invisible. As a result only the text object remains visible while the parent is not. *basicVisi_1* is set to *False* when user selects an intermediate or advances tutorial option and is set to *True* when basic is selected. Value of *animationProp_1* variable controls the animation shown while clicking over it in a similar fashion as described in previous paragraph. Similarly other options under basic type have been created. *Logic Editor* panel of basicScreen is shown in figure 6.3

2. **Designing exercises**

In the practice exercise scenes *Logic Bricks* have been used along with striker, coin and various other objects. Let's talk about the striker's logic bricks first. There are two 'Always' type sensors connected with two different 'Python' controllers. One of them is controlled with *striker_straightShot_1.py*. Inside this script it is checked if the required conditions are met or not after user plays a shot. If it does not met then 'Scene' actuator connected with it reloads the same scene(here straightShot_1 scene, as Straight shot skill is still being considered as example to describe the scenario). There is a 'Keyboard' sensor which detects whether space-bar is pressed or not. If space-bar is pressed then it sends a positive pulse for a very short duration. This

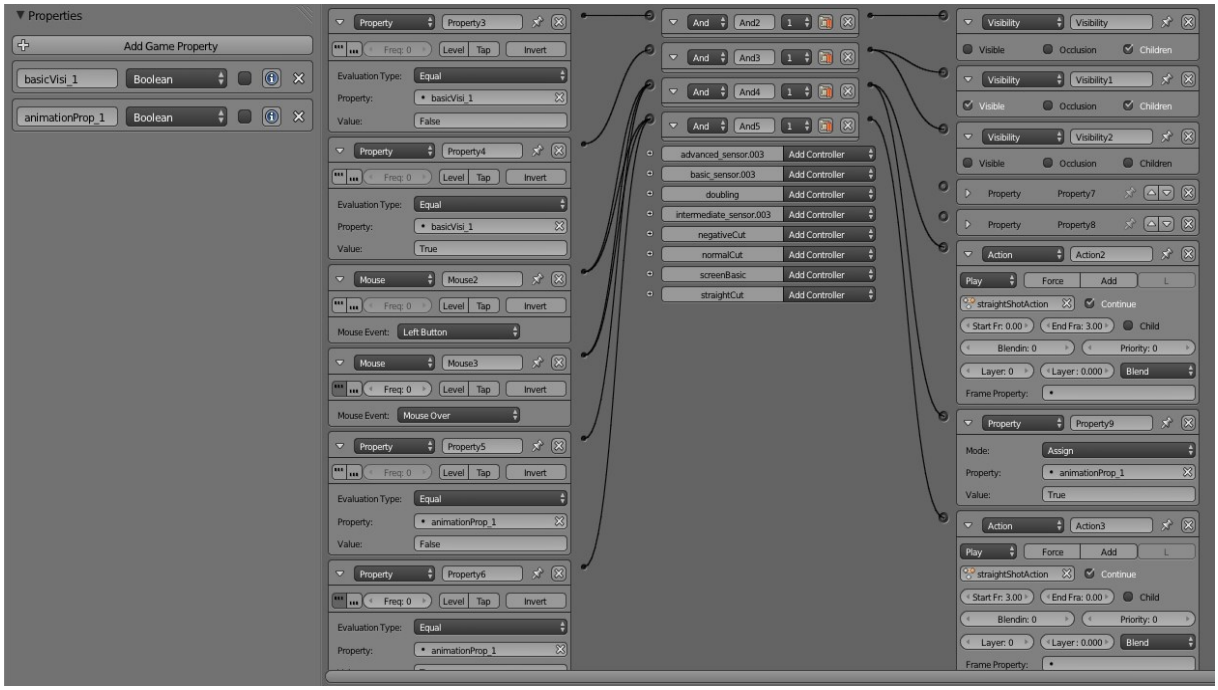

Figure 6.3: Logic bricks for basicScene

functionality is achieved by setting the *Tap* property this sensor. This sensor is connected with a 'Scene' actuator via an 'And' controller. If space-bar is pressed then control goes back to basicScene. Please refer to the figure 6.4.

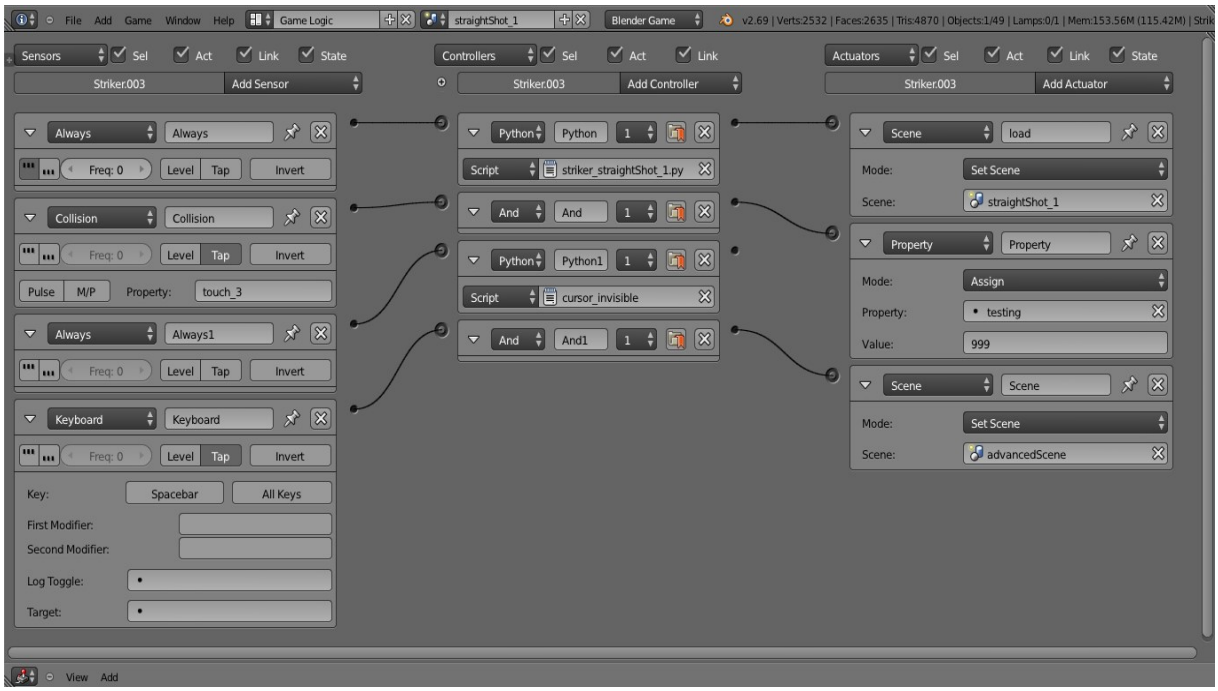

Figure 6.4: Striker's logical components for practice exercise

Several sensors, controllers and actuators are used for the coin in starightShot_1 scene. Initially value 100 is stored in a global variable 'health_coin1' . There are four 'Collision' sensors to detect whether the coin has collided with any of those invisible spheres positioned at pockets of the carrom board surface. Actually it is being detected whether the coin has collided with the properties of those spheres or not. In *Blender Game Engine* collisions can be detected between an object and some properties of another object rather than between two objects. These 'Collision' sensors send out a positive pulse for a short duration when the collision occurs with the sphere properties(namely *point_1*, *point_2*, *point_3* and *point_4*) as their *Tap* property is set. These four sensors have been connected with an 'Or' controller. This controller connects to an 'Action' actuator. Now here is a little trick. As the coin collides with a sphere the Z-axis location of the coin is decremented by 2 *Blender units* to smooth out playing experience. It is done by setting the motion type as simple motion and setting the Z-axis value to -2 and setting the L(stands for local) property of 'Action' actuator(figure 6.5). Next we are detecting the collision between the coin and the planes underneath the pockets. This is done in the similar fashion described before. After the coin is collided with any of those planes 50 is deducted from 'health_coin1' . Later we are checking if a coin is pocketed in a particular hole or not using it's 'health_coin$i$' value. Also collision(s) with the four borders are detected for some other use. *Logic Editor* panel for coin is shown below in figure 6.5. All logical components in figure 6.5 were not expanded. Unexpanded sensors have same structures like the expanded one just above them.

Complex exercises listed in exerciseScene are somewhat big and consist of some intermediate steps. When user pockets one coin then the resulting scenario after pocketing one coin should be represented to user. Let's consider about coin 1. It is desired that user pockets it at the top left hole. So after the coin collides with the *hole_1* the positive pulse is feed to a 'Game' actuator. Setting the option to *Save bge.logic.globalDict* in this actuator does the state save. It saves the overall scenario of the whole game into a global dictionary when it is triggered. After the state is saved into the global dictionary we need to load it to recreate that exact scenario. An 'Actuator' type sensor is used. This type of actuators give positive pulse when some certain actuator gets activated. Here it senses whether the actuator related with saving state into global dictionary is triggered or not. If yes then it loads the content of global dictionary using the 'Game' actuator associated with it. Scene can also be restarted using 'Scene'actuator. After user successfully completes the exercise control will be redirected to exerciseScene. This is shown below in figure

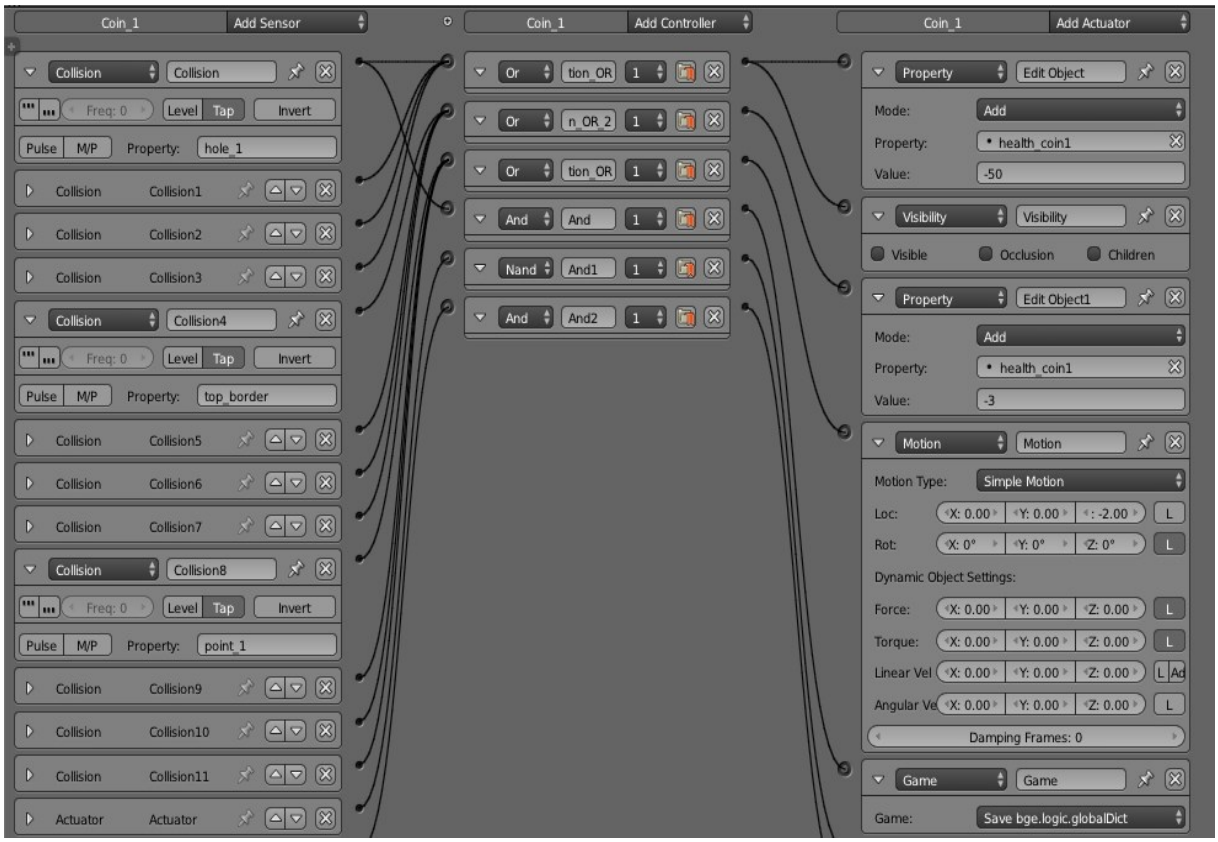

Figure 6.5: Logic bricks for coins

6.6.

Logic editor panel for ***Dependencies between intra-Scene objects*** & ***Displaying videos and presenting practice exercises*** are presented in dissertation written by Mayur Katke[9].

6.3 Python Scripting

Python is used as scripting language in *Blender game Engine*. This game engine provides a very interactive layer for creating and editing python scripts. Usage of Python programming enhances the functionalities and vastness of *Blender*. All game object properties can be manipulated using python scripts. Python scripts manipulate *Blender*'s data in same fashion as the UI and other layers do.

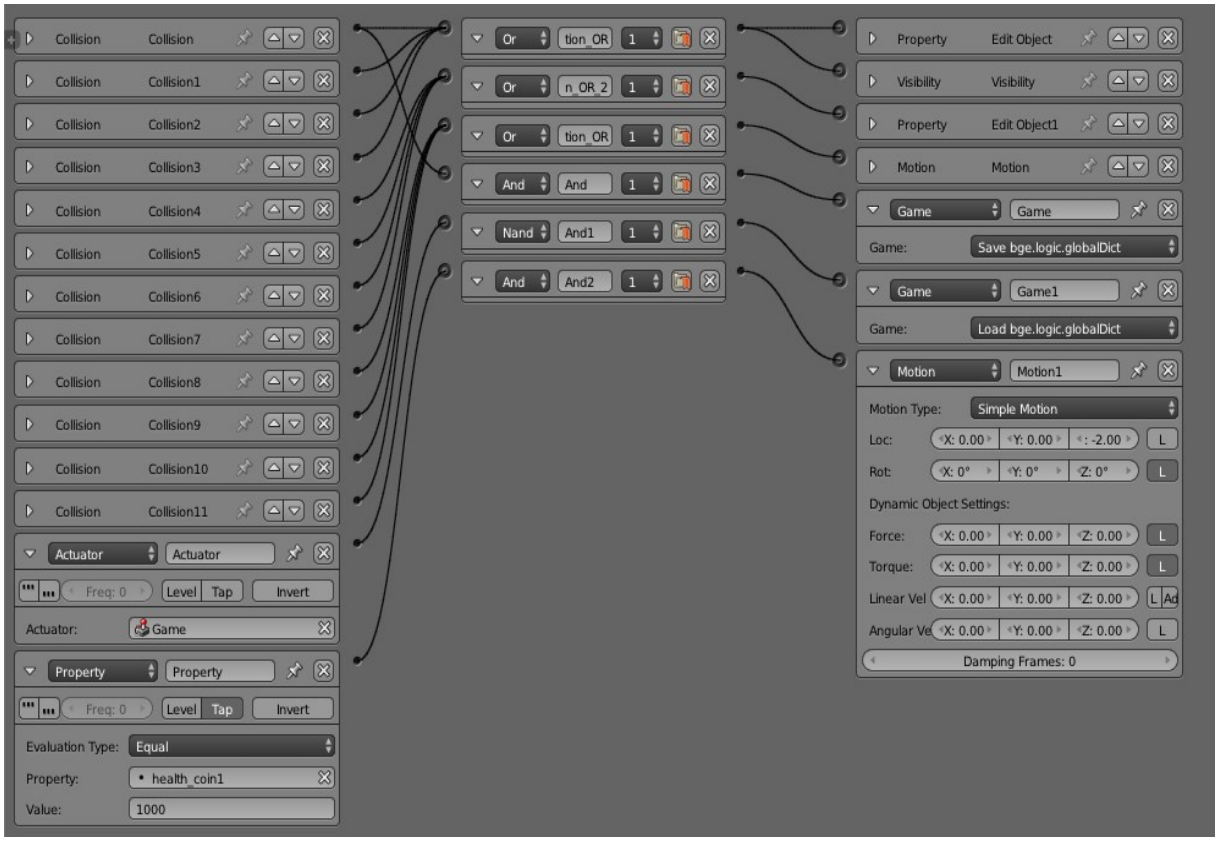

Figure 6.6: Usage of global dictionary in logic bricks

Blender provides a python scripting layer called "Text Editor". This editor is very interactive. It makes creation and testing of a .py script very easy. We can test the correctness of a python script just by pressing the 'Run Script' button in the *Text Editor* panel. Python scripting in *Blender* provides the freedom to control game objects and get the desired functionalities from them. Scripts are used in "Python" controllers in *Logic Editor* panel. Sensor components in *Logic Editor* feed positive pulses to these "Python" controllers, when triggered. These controllers run the scripts attached with it after getting positive pulses. Actuators can also be connected with these type of controllers to perform specific actions.

To accomplish the desired features of the system, *Carrom Tutor 2.0*, python scripts were used upon game objects like striker, coin, empty object and text objects. This is done in almost all of the scenes. Many scripts were written and two scripts were downloaded from blender communities to enhance the tutor. Some main modules of these scripts are given below.

1. **Power slider**

After user clicks for the first time during any exercise, power slider gets activated. It moves up and down between its highest and lowest value continuously. The power slider is made by using ten mesh objects of type *'plane'*. The height of the power slider is varied by showing different numbers of planes with respect to time. This visualization of varying number of planes is done in a way so that they follow *sine wave* characteristics. A variable "timer" is used to make those planes visible. This power slider is attached to the camera so that it moves according to mouse movement. Python code for the first three slider planes is given on next page.

```
if ((striker["slider_enabled"]) and (not striker["striker_released"])):
    half_time_period = 70
    time_period = 2 * half_time_period

    striker["timer"] += 1
    if(striker["timer"] == time_period):
        striker["timer"] = 1
    v = (sin(3.141592*(striker["timer"]/half_time_period) -
   1.570796)+1)*500

    if(v > 0):
        striker["slider_1"] = True
        striker["force_multiplier"] = 5
    else:
        striker["slider_1"] = False
    if(v > 100):
        striker["slider_2"] = True
        striker["force_multiplier"] = 10
    else:
        striker["slider_2"] = False
    if(v > 200):
        striker["slider_3"] = True
        striker["force_multiplier"] = 15
    else:
        striker["slider_3"] = False
```

2. **Displaying videos**

In tutorial scene, user can see the list of basic, intermediate and advanced skills. A video showing how to play a particular skill is shown to user when he clicks on button for that skill. After video, two board situations are given to user one by one for applying the same skill. Firstly various properties of screen, which is actually a plane object, were set properly to nullify malfunctioning. A python controller is there for every skills in list to play the video using script. This script is downloaded

from *www.tutorialsforblender3d.com*[3]. This script gets the owner of the controller on which it is applied. Then it changes the material of that object according to the value stored in the variable used for material(for an example "materialBasic" in case of basic skills). Path of the videos corresponding to each skills have been stored in different global variables named "movie$i$". This script plays the respective video when a button is clicked by finding the file using the path variable. There is an optional boolean variable used named "loop". If it's value is *True* then the video gets played repetitively during the given delay. It is set to *false* in this case.

3. **Scoring in exercises**

Each exercise given in the tutor tests how well user can apply the skills. Evaluation is present in the system. Complex exercises have a total point of 100 to be scored. In these exercises user has two or more black coins to pocket without giving opponent a chance. Score given to user depends upon the shot chosen. If optimal shot is chosen then user is awarded with maximum points. This optimality of shot selection is done by detecting collisions among striker, coins, border of board. Each coin has a property "health_coin$i$"(where i is the coin number assigned for implementation) which is set to 100 initially. When finally a coin is pocketed then this value is changed to 1000 so that points can be assigned to *score* object. A snippet of code is given below which handles the pocketing of coin1 and coin3 in Exercise_1 and assigns points.

Python code snippets related to ***Changing users view according to mouse movements***, ***Showing strikers path as a line***, ***Strikers orientation and applying force*** have been given in dissertation written by Mayur Katke[9] with explanation. Script named ***"mousemove.py"***[13] captures mouse movement functionalities and applies that on camera. Regarding this blender community[1] was also helpful. This script has been downloaded and used from blender community[13]. Details have been given in dissertation written by Mayur Katke[9].

```
score_text.text = str(score_text["points"]) + '/100'
if (coin1["health_coin1"] == 1000 and own["flag1"] == False):
    if (coin3["health_coin3"] != 1000):
        if (coin2["touch_coin2"] != 0):
            score_text["points"] += 75
            own["flag1"] = True
        else:
            score_text["points"] += 40
            own["flag1"] = True

if (coin3["health_coin3"] == 1000 and own["flag2"] == False):
    score_text["points"] += 25
    own["flag2"] = True
```

Chapter 7

Demonstration

Carrom Tutor 2.0 starts with *startScene*, which gives three options to users; Start, How to and Quit. "Start button" loads the second scene with other options. "How to" button displays some instructions about playing shots in exercises. Figure 7.1 shows the startScene with instructions.

Figure 7.1: Start screen

Figure 7.2 given below shows the second scene loaded after clicking on start button. There are four buttons in this scene, viz. Tutorial, Exercise, Strategies and Game rules.

The corresponding scenes will be loaded after clicking on these buttons.

Figure 7.2: Second screen

The tutorial scene has three buttons namely Basic, Intermediate and Advanced. Carrom skills are divided in these three categories and corresponding list of skills is displayed after clicking on these buttons. Figure 7.3 shows the tutorial scene with list of advanced skills displayed in it. If user clicks on any of these skills, the corresponding video of skill is shown in right portion of figure 7.3. This video gives instructions and a demo of skill to user. After watching this video user is automatically directed to corresponding practice exercise scene in which user can try to play the same shot from the video. When user plays this correctly, second practice exercise scene gets loaded where user can apply same skill.

Exercise scene has the list of exercises provided to users for testing their skills. User is directed to the corresponding exercise scene after clicking on any button. Each exercise is a complex board situation. User must pocket her coins without giving opponent a chance to get maximum points. Score out of hundred is shown at the right corner of scene. It is updated after each shot played by user. If user does not pocket all coins then same exercise scene is reloaded for playing. Second exercise is shown in figure 7.4.

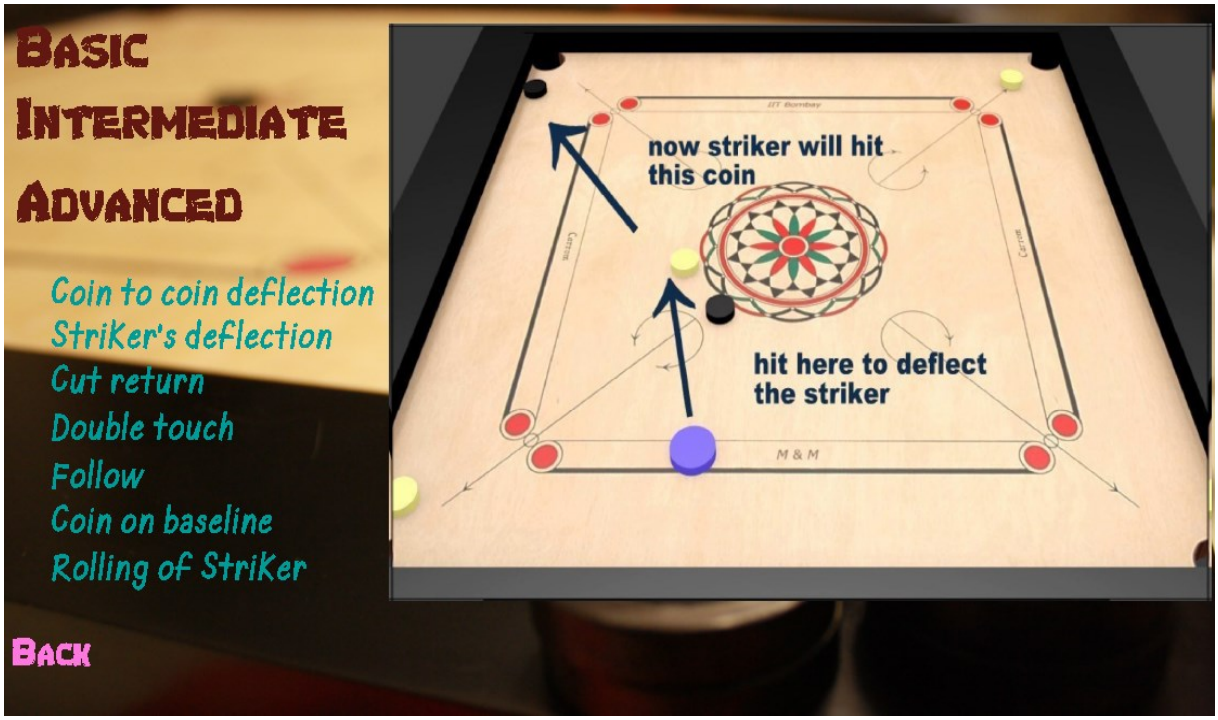

Figure 7.3: Tutorial screen

Figure 7.4: Screen shot of second exercise

Carrom board game strategies for singles and doubles are listed in the scene "Strategies". Also carrom game rules are given in scene "Game Rules". Links for these scenes are provided in second scene.

Chapter 8

User Experiment

Description of user experiment done for *Carrom Tutor 2.0* has been described in this chapter. Firstly the sample ,i.e. desired audience, then data collection procedure has been mentioned. Later analysis of the collected data has been given.

8.1 Sample

Any one who has a basic knowledge about carrom and played carrom game few times can participate in the user experiment. Eleven users participated in *Carrom Tutor 2.0*'s user experiment. These group of users were mixture of beginners, intermediates and experts in carrom.

8.2 Data Collection Methodology

Pre-test was also conducted for *Carrom Tutor 2.0* by telling the user to play complex exercises prior to the exposure with the tutorials and corresponding practice exercises. After going through complex exercises user was advised to experience the whole tutor step by step starting from tutorials then practice exercises and in the end complex exercises. This was done to check the level of difficulty faced by user while going through complex exercises with and without the help of tutorials and practice exercises. Additionally each user was given an online carrom application, ***Carrom King***[10] to play with. This application is similar to other carrom games available online. Later users were asked to

compare between *Carrom Tutor 2.0* and *Carrom King*. Half of the users were asked to play with this *Carrom King* application before interacting with *Carrom Tutor 2.0* and rest of them were instructed to do the opposite.

Users were asked to give answers to the following questions given in a form after she is done with the experiments.

- User's exposure to Carrom
- How many new Carrom skills have you learnt?
- How was it playing the exercises before watching the demos?
- How was it playing the exercises after watching the demos?
- How diffucult was it to relate the demos with the shots needed to play?
- Compare with other Carrom application you played with

The data collected so far was for checking the learning gain. For usability testing of the system, *SUS (System Usability Scale)* analysis was done using [2, 4]. In this analysis, five point likert scale questions were asked to users. These questions were asked for checking accessibility, efficiency, effectiveness and attractiveness of the user interface of system.

8.3 Data Analysis and Results

SUS analysis was done to check usability, attractiveness, efficiency of *Carrom Tutor 2.0*. Details about the questions asked in *SUS* and their responses are discussed in dissertation written by Mayur Katke[9]. Average percentage of SUS score for our *Carrom Tutor 2.0* is **84.09**. A SUS score of 68 is generally considered as average. *Carrom Tutor 1.0* has an average score of 77.14(grade B). *Carrom Tutor 2.0* has an average score of 84.09(grade A). According to the average SUS percentage *Carrom Tutor 2.0* also has surpassed it's predecessor. It can be inferred that user flexibility, interface design, content design play important roles in a tutor design. Figure 8.1 given below plots average rating of each question.

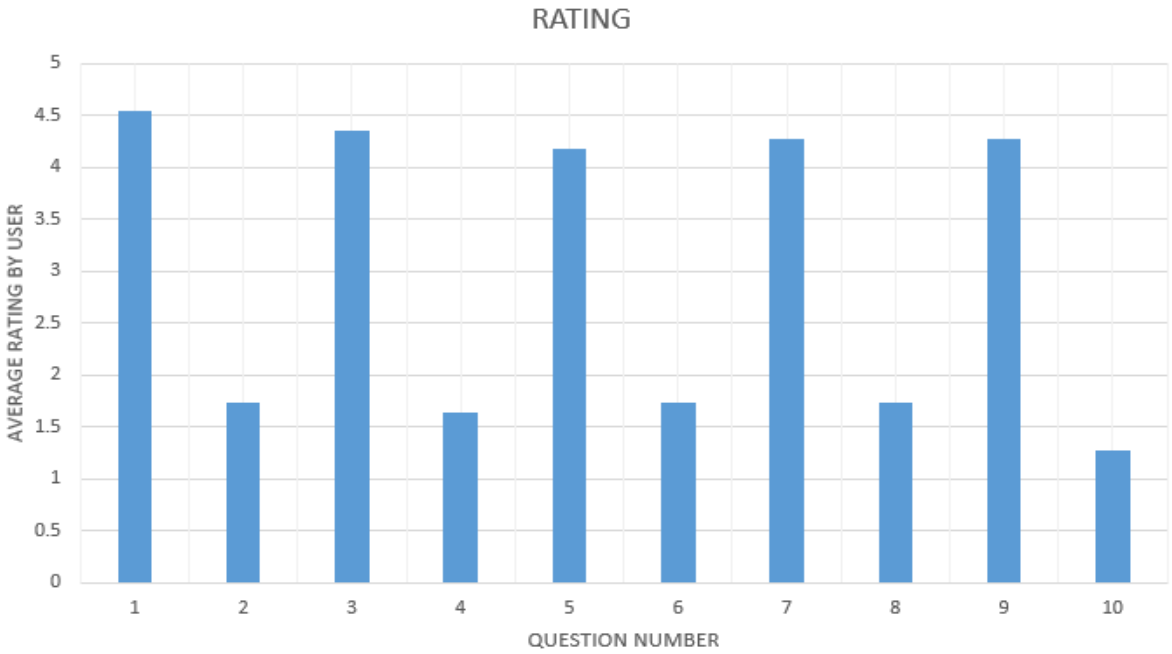

Figure 8.1: SUS analysis of *M & M Carrom Tutor*

Now let's think about the reliability and validity of this result. It has been seen that *SUS* provides reliable result[15]. *SUS* questions are arranged in a very proper way. Few questions are also repeated but asked from a different angle. These types of questions help reducing the effects of random answers given by people. It has also been observed that *SUS* can produce reliable data from a very very small set of users. Validity is how well the usability of a system is measured by *SUS* analysis. *SUS* can efficiently measure the effectiveness, attractiveness of a system[15].

Following information has been gathered from the form mentioned in 8.2. All users stated that playing the complex exercises without interacting with tutorials and practice exercises was difficult. Similarly it has been mentioned by the users that complex exercises seemed easier after watching the tutorials and practice exercises. When asked about how many new carrom skills they have learnt after using the tutor, answers vary in between 3 and 6. Relating the demos with the exercises were easy for the users too. So it is prevalent that learning is happening while user interacts with the tutor. Users also provided very good and motivating feedbacks to our tutor compared to other carrom applications available online. Essence of those feedbacks is listed below,

- very interactive tutoring system

- presence of 3D/user view is an added advantage
- interface(GUI) is far better compared to *Carrom Tutor 1.0* and other applications
- feel of a real carrom board while playing the exercises
- other applications does not provide any tutoring so this tutor encourages learning
- rotation of striker with mouse movement is very good
- line to draw striker's path is very useful for smooth handling of striker

Chapter 9

Challenges

After *Carrom Tutor 1.0* it was needed to build a better system which will provide more flexibility and control to user. The experience while interacting with it has to be more vivid. So in order to give more flexibility to user experience it is needed to create a game-like environment. For this purpose *Blender Game Engine* was used. Learning the functionalities of a game engine is itself a challenge. It took substantial amount of time to get used to this game engine. Initially some basic modelling was tried to get a feel of how the objects can be transformed, what are the different property types and their functionalities etc. First step of making *Carrom Tutor 2.0* was creating the carrom board. Cutting four holes(pockets) from a cubical mesh object was some what tricky. Boolean modifier was needed to do this difference operation. After the carrom board surface is made an unexpected problem arose. Coins(cylindrical objects) were not passing through those holes though the holes were created properly. After a lot of experimentation finally it was found that there is an issue with the bounding box of the carrom surface. As the object created for carrom board was a cube so its bounding box was also having the same shape of a cube. Even if after doing the boolean difference the object's shape had been modified but the bounding box remained same. Though the board surface was changed as it should have been but there was an invisible bounding box which was not letting the coins to pass through. Another problem was there regarding the carrom board surface object. Boolean difference operation also modified the surface near to those pockets a bit. It created a more complex mesh around the holes resulting the coins and striker to bounce while in motion. A different approach was needed. Later the carrom board surface was made by three cubic mesh objects. Holes were created by applying boolean difference operation on small cubic mesh objects with cylindrical mesh object where circular diameter was equal with sides of those small cubes.

Though *Blender* takes care of applying physics laws on objects but setting them up properly is difficult. Firstly, the type of objects are needed to be set. Moving objects like striker, coins are set as *dynamic* object. Dynamic object follows the laws of physics. Additionally python scripts can also be used to manipulate the behavior of these *dynamic* objects. Carrom board surface, border etc. are *static* objects. *Static* objects generally remain at a fixed position. Next is setting different parameters of these objects so that object behavior matches with reality. Parameters like friction coefficient, damping coefficient, elasticity etc. were very crucial for real-like visualization of carrom shots. Different set of values of these parameters were tested. After many trial and error testing, a set of values was finalized. This finalization of values was done completely based on intuition and experience of playing carrom. This set of values gave more realistic result among the tried sets of values.

Logic Editor is one of the integral part of *Blender Game Engine*. But at the beginning it was troublesome to apply it properly. There are many different types of sensors, controllers and actuators available as logical components. To understand what each of these components does took a significant amount of practice. Each component also has some different properties of it and altogether it makes some what complex scenario during implementation. As an example if basicScene is considered, one can see the numerous interconnections between different logical components. Implementation in *Logical Editor* gets more and more complicated as the required functionalities of a scene increases. Initially it was hard to create the interconnections among intra-scene objects due to lack of knowledge. At first exercises were created as different *.blend* files. External *.blend* files can be invoked in a different *.blend* file, but there is a problem if in-game save/load is required. If the external *.blend* file terminates then the control does not return back to the *.blend* file from which it is called. So this created a problem implementing the system. Later it was decided to create scenes in one *.blend* file rather than creating and calling many external *.blend* files. Saving/loading is performed using the 'Scene' actuators and using python scripting. Proper usage of the global dictionary to save and load the situations also required some practice.

One of the most challenging part in creating *Logic Bricks* is managing numerous connections in a relatively complex scene. If the basicScene is taken for an example, there are numerous connections established between logical components. In basicScene there is a screen object. *Property* sensors of five basic skills are connected to it's five different python controller. There is one *delay* sensor and one *property* sensor for each basic

skills that are connected to five different *and* controllers. Those *and* controllers are also connected to different *scene* actuators. For each basic skill there are four *property* sensors and two *mouse* sensors that are connected to three *visibility*, two *action* and three *property* actuators via four *and* controllers. Basic, intermediate, advanced plane objects have four sensors, two controllers. These are connected to actuators of each basic skill. Approximately in basicScene there are 150 logical components and 150 logical connections in total. It becomes very confusing to handle these many logical connections in a single panel. There are 46 scenes in *Carrom Tutor 2.0*. Images of *Logic Editor* panel for basicScene is given in figures 9.1, 9.2, 9.3.

Building this tutor was a great experience altogether. It prized the knowledge of how a game engine is used to make a game and the versatility of its power.

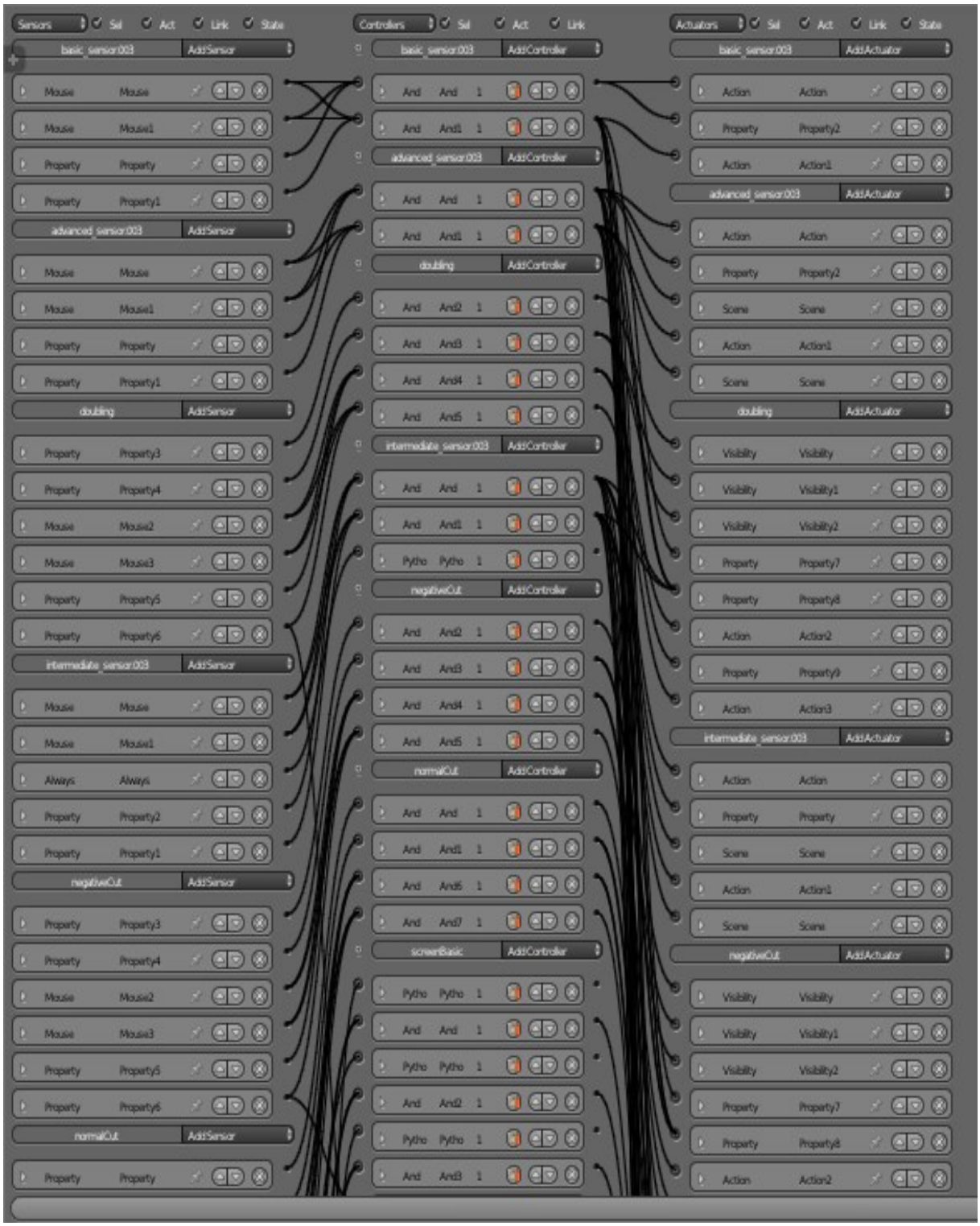

Figure 9.1: Logic bricks for basicScene, part 1

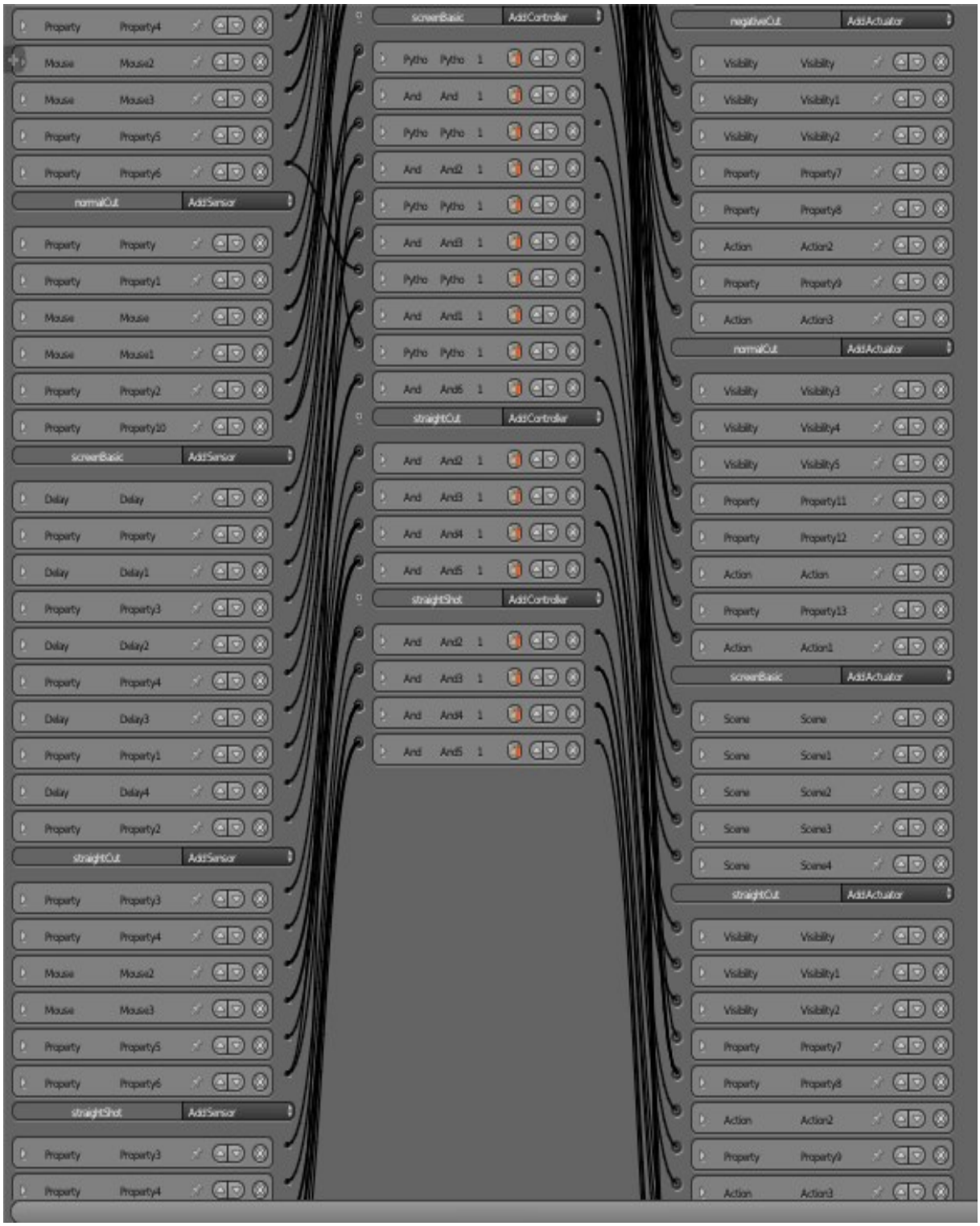

Figure 9.2: Logic bricks for basicScene, part 2

Figure 9.3: Logic bricks for basicScene, part 3

Chapter 10

Conclusion and Future Work

10.1 Conclusion

Main aim of *Game Based Carrom Tutor* is to increase knowledge about carrom game in people. As mentioned earlies there is no such game based carrom tutor available on web. *Carrom Tutor 1.0* was the first initiative towards achieving this goal. This system is actually a web application, which teaches various carrom skills to users. It also tests user's learning through exercises. Though it was not robust when user flexibility, attractiveness etc. are considered. According to it's usability study, it can be seen that irrespective of having some shortcomings this system performs well when learning is considered. To overcome the shortcomings of *Carrom Tutor 1.0*, *Carrom Tutor 2.0* was built. *Carrom Tutor 2.0* is very much interactive. User can place striker at any place along the base line, can execute any shot according to her wish with desired force. This is what people do in actual carrom board game. One places striker at a suitable position along the baseline, thinks about which shot to perform and then play the shot with appropriate force. Also the demos of various carrom skills were redesigned to offer a better feel and also to increase the scope of learning. *Carrom Tutor 2.0* has better user feedback, higher *SUS* score etc. If game based learning is considered then it can easily be seen that design of a system plays a very important role. The system should be interactive, interesting, attractive, diverse. Strategies to present the system to user play essential roles in it's success.

10.2 Future Work

Presently the number of complex exercises given in the tutor is less. It is expected that more exercises will be added very soon.

There are few unexpected behavior experienced in *Blender* which are yet to be dealt with. One of these is related to collision bound. This problem introduces some unwanted behavior in some objects. Solving these issues will make the tutor run smoother in future.

It will be beneficial if a full carrom game against artificial intelligence can be incorporated with this tutor. Having this feature will surely enhance the scope of applying learnt skills.

Bibliography

[1] Blender community. http://blenderartists.org.

[2] Sus analysis application. http://www.mohini.0fees.net/iit/welcome.php.

[3] Tutorials For Blender 3D. http://tutorialsforblender3d.com.

[4] John Brooke. *SUS - A quick and dirty usability scale*. Digital Equipment Corporation, 1986.

[5] Allan Collins. *Cognitive Apprenticeship*, chapter 4, Handbook of the Learning Sciences. Cambridge Univ. Press, 2006.

[6] Paul J. Diefenbach. Practical game design and development pedagogy. *Published by IEEE Computer Society*, pages 84–88, May/June 2011.

[7] Blender Game Engine. http://blender.org.

[8] Richard Wainess Harold F. O'Neil and Eva L. Baker. Classification of learning outcomes: evidence from the computer games literature. *The Curriculum Journal*, 16(4):455–474, December 2005.

[9] Mayur Katke. Carrom tutor : Playing strategies and impementation. Master's thesis, IIT Bombay, June 2014.

[10] Carrom King. http://twoplayergames.org/play/719-Carrom_King.html.

[11] Imgrad Schinnerl Maja Pivec, Olga Dziabenko. Aspects of game-based learning. In *Proceedings of I-KNOW '03, Graz, Austria*, pages 6–20, July 2003.

[12] Iván Martínez-Ortiz José Luis Sierra Baltasar Fernández-Manjón Pablo Moreno-Ger, Daniel Burgos. Educational game design for online education. *Computers in Human Behavior*, pages 2530–2540, 2008.

[13] Riyuzakisan. 3d art - game design. http://riyuzakisan.weebly.com.

[14] Tutor. http://en.wikipedia.org/wiki/Tutor.

[15] Measuring usability. http://measuringusability.com/sus.php.

[16] David Rosenthal Viswanathan Kodaganallur, Rob Weitz. Tools for building intelligent tutoring systems. *Proceedings of the 39th Hawaii International Conference on System Sciences - 2006*, pages 1–10, 2006.