Introduction

Pool originally known as Billards is a game invented in the 15th Century is a game of rich history "The game has been played by kings and commoners, presidents, mental patients, ladies, gentlemen, and hustlers alike. It evolved from a lawn game similar to the croquet played some-time during the 15th century in Northern Europe and probably in France." (BCA, 2014) Since then Pool has become a Worldwide recognised sport, played by millions around the World "there are about 46 million people playing pool in about 9,000 facilities throughout North America. Ninety-five percent of those people play pool. Only three percent and two percent play carom and snooker respectively." (College Foundation, 2017)

Rules of The Game

The objective of the game is to pocket a specific group of balls before the other player "The object of the game is to win by being the first player to Pot a group of Colours in any order and in any pockets and then Pot the Eight-Ball in any pocket." (EPA, 2009)

In terms of the legal objects within the game, pool is played with a total of fifth teen balls, these are divided into two groups and each player is assigned a group "Balls comprise of two numbered groups, 1-7 which are solid coloured balls, 9-15 are striped coloured balls, the 8 ball is a solid colour black." (EPA, 2009)

The choice of ball colours is made once the opening shot has been made and the first legal pocketed ball has been completed of a set colour "On the first occasion a player legally pockets an object ball, including following a foul, then that ball denotes their group, unless one or more of both groups are pocketed, the player MUST then nominate a group before play continues." (EPA, 2009)

The Black ball or more commonly known as the 8 ball, is the game winning or losing ball. In terms of winning the game, this can be achieved once all of the player's colour balls have already been pocketed "The player or team pocketing all their group of object balls in any order, and then legally pocketing the 8 ball, wins the game." (EPA, 2009) thus alternatively if a player pockets the black ball while colour balls of their group are still on the table, loses the game "If a player pockets the 8 ball (black) before all the balls in their own group, except on the break, the player loses the game." (EPA, 2009)

Description and Solutions (Functionality, Classes)

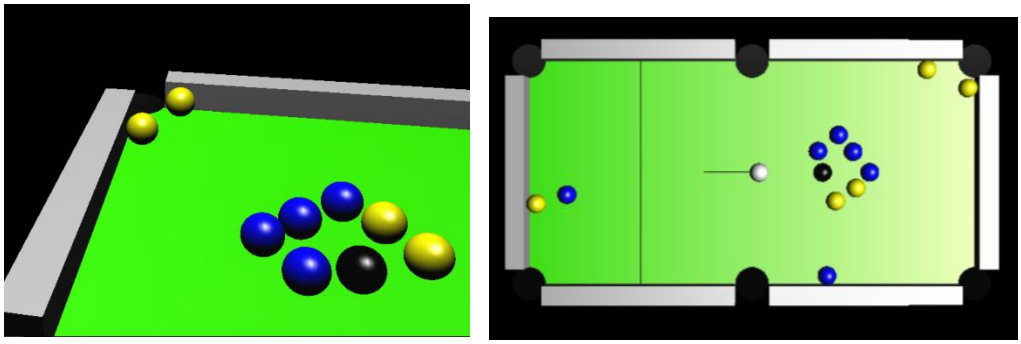

Figure 1: Pockets

Pocket Feature

This assignment involved adapting a pre-existing game of pool from the workshops and implementing a number of additional features. The first main feature implemented was the Pockets. Therefore, a separate class was created to house this feature "PocketTable.cpp" along with the main header file "simulation.h" The first process to creating the feature was defining in game variables,

this included the number of pockets, in this case following the traditional rule of Pool is six Pockets, then also defining the radius of each Pocket. Once the header has been completed the next important feature to develop was the ability to check if a ball has come into contact with the pocket, this was achieved by checking the boundary of the pocket "`position(0) >= poc.position(0) - poc.radius`" Thus if the ball is below a certain distance, it will returns true, this will then call upon a response function "`Hit_Pocket()`". In relevance to the response, the program must first check if the ball that was pocketed was legal, thus checking if the white ball has been pocketed or the black ball has been pocketed, this can be achieved as all balls are given an integer value "`if (colour == 0)`", this will then result in a change of player. Alternatively, if a ball has been pocketed legally this will result in the ball being taken off the table "`position = vec3<float>(500.0f, 0.0f, 0.0f);`" and its velocity being set to zero "`velocity = vec3<float>(0.0f, 0.0f, 0.0f);`" Once completed the program will recount the balls on the table. Furthermore, if a player has not yet been assigned a colour group of ball this will then activate the function to assign a specific player the pocketed colour, this will depend upon the integer value of the ball that has gone within the boundary of a Pocket "`if (gTable.players[gTable.CurrentPlayerCount]->BallColourValue == 0)`" Lastly the pocket response must also look if the black ball was pocketed legally "`((gTable.players[gTable.CurrentPlayerCount]->BallColourValue == 2 && gTable.numBlueBalls == 0)`" this will result in the player winning the game, alternatively if a player pockets the black ball with colour balls still on the table, the player will lose, which is a simple else statement on the above statement.

The last main implementation of Pockets was its design implementation. In terms of each pocket position there needs to be a set location for the program to understand its location, thus the pre-existing cushion class can be used a guide point to calculate the effective positions of each pocket "`pockets[0].position = cushions[0].cushionFaces[0].vertices[0] + vec3<float>(0.0f, 0.01f, -gTable.pockets[0].radius);`" Once this has been implanted the "PoolGame.cpp" can call this information.

Player Feature

The next main component of the pool game implemented was the player feature this allows for the game to be played by multiple people at the same time, this was again given its own class "Player.cpp" with a connection of the header file "Simulation.h". The first process to creating this feature was to define the in game variables, in this case this includes the number of players. Furthermore, initial player variables also include the number of shots the player has left, their ball colour set assigned, if they have potted the black ball. Once these variables have been initialised the main components of player were then built. Firstly, the program needs to initialise both players with equal starting variables "`PlayerShotsLeft = 0;, BallColourValue = 0;, HasBlackBeenPotted = false;, HasPlayerHitABall = false;`" Thus making the game fair. Once completed the game also needs to know whether the player needs to be switched, this is calculated by checking if the player has used the cue and made any illegal moves, as seen in figure two, if the player hits the black ball first, this will result in the other player gaining an extra turn and a swap of players.

```
285  void ball::HitBall(ball &b)
286  {
287      // White Ball (Cue Ball) No Colour has been Set Yet
288      if (CountBall == 0 && !gTable.players[gTable.CurrentPlayerCount]->HasPlayerHitABall)
289      {
290          gTable.players[gTable.CurrentPlayerCount]->HasPlayerHitABall = true;  // Player has
291
292          switch (gTable.players[gTable.CurrentPlayerCount]->BallColourValue)
293          {
294          case 0: // White Ball (Cue Ball)
295          {
296              if (b.colour == ball::Black) // If Black Ball has been Hit First
297              {
298                  gTable.ExtraTurn = 1;           // Other User Gets Extra Shot (2)
299                  gTable.ChangePlayer = true;     // Changes Player with Extra Shot
300              }
301              break;
302          }
```

Figure 2: Illegal move, change player

Furthermore, this class functions alongside that of the ball and pocket classes which send either legal or illegal parameters back to the class thus deciding whether the player should get an extra turn or the player should be changed. In terms of the game, player one is responsible for the first turn, which involves breaking the rack (see figure 3) This will then allocate a colour if the move is considered legal. The main processes of this feature are repeated until one player wins the game.

```
86
87    // inital player setup
88    void table::SetupPlayers(void)
89    {
90        for (int i = 0; i < NUM_PLAYERS; i++)
91        {
92            players.push_back(new Player());
93        }
94        if (CurrentPlayerCount == 0) // Player Number One
95        {
96            NextPlayerCount = 1;
97            players[CurrentPlayerCount]->Reset();
98            players[NextPlayerCount]->Reset();
99            players[CurrentPlayerCount]->PlayerShotsLeft = 1;
100       }
101       else if (CurrentPlayerCount == 1) // Player Number Two
102       {
103           NextPlayerCount = 0;
104           players[CurrentPlayerCount]->Reset();
105           players[NextPlayerCount]->Reset();
106           players[CurrentPlayerCount]->PlayerShotsLeft = 1;
107       }
108       ChangePlayer = false;
109   }
```

Figure 3: First Move

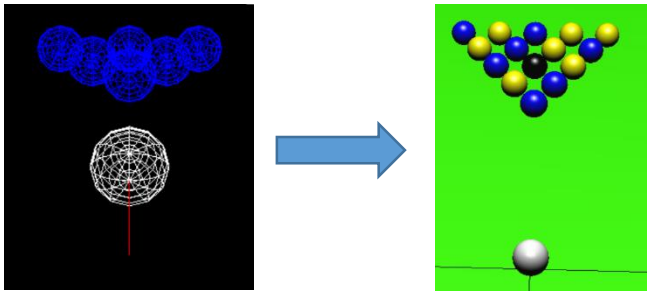

Figure 4: Wireframe to Solid Colour Transformation

Colours (Solid)

In terms of the pre-existing game given within the workshops, a lot of the main functionality was already present, although what was not present was many of the design features. One of the main design flaws of the original piece was that balls and objects were all working under a wireframe model, this seemed quite boring and unprofessional, thus a solid colour transformation was implemented. This implementation occurred to both the balls, cushions and table floor. The first process to implementing this relied on the pre-existing light feature being active as without this feature wireframe was only possible, thus this meant checking the light source is on within many functions. Secondly a solid circle function had to be implemented, this was possible by using the "`GL_TRIANGLE_FAN`" This allows for vertices to all be connected with their previous counterpart, thus allowing for a common vertex, to which can then be used as a surface, thus this meant that a solid circle can be produced to which in this case can be used to create a black circle for the pocket.

In relevance to the colour of the balls, this can be achieved by using the "`glMaterial`" parameter, thus this allows the ability for a float value colour to be assigned, in this case each ball has an integer value, thus each ball can be assigned a colour through the use of an IF statement, with balls integers for each case.

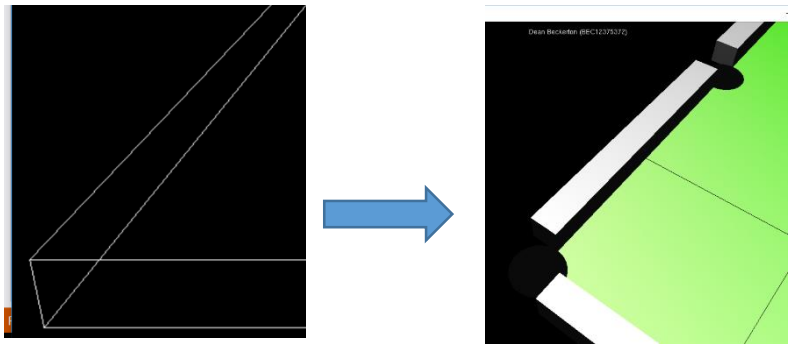

Figure 5: Dimensional Cushions (3D)

Vertices (Cushions 3D

In terms of simplicity this upgrade was quite simple to implement. Firstly, this involved changing the pre-existing cushion class vertices, from this it then involved changing the normal function of the cushion class from simply calculating just two dimensions "`vec2 temp = vertices[1] - vertices[0];`" to instead taking into account four dimensions and then output the final value "`vec3<float> temp = vertices[2] - vertices[1];vec3<float> temp2 = vertices[0] - vertices[1];`" Overall this simple change arguably had a dramatic change to the design quality of the game.

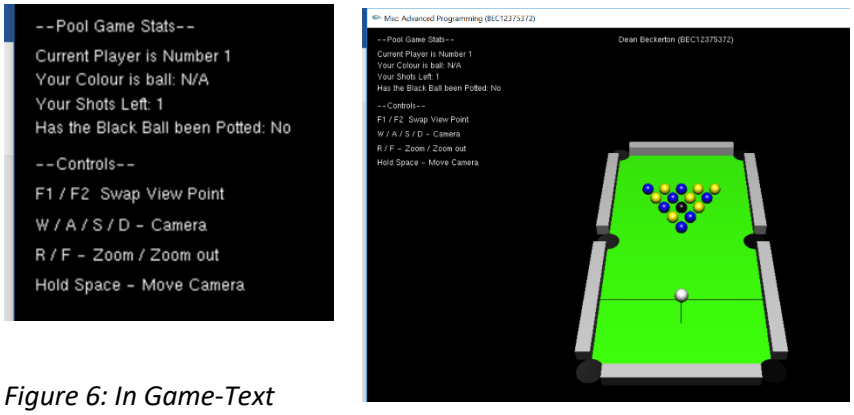

Figure 6: In Game-Text

Text Display In-Game Feature

Since within the brief it stated this was not a requirement, it seemed like an extra feature could be to implement text over the game display. This was again an arguably simple feature to implement. This firstly involved creating a function to set the properties of the text, this was completed within the "PoolGame.cpp" file. This function is known as the "display_text" function, within this function three main opengl components were used glRasterPos3i, this specifies the raster position for pixel operations, this means the position of the display of text, the next component includes the GL_PROJETION, this works similar to that of a stack, this allows for subsequent matrix operations to the projection. Lastly glOrtho is used which multiplies the current matrix by an orthographic matrix. Thus this allows for the ability to display text over the previous matrix. The last main process involves the actual text displayed. Displaying the specific text was quite simple "`Dispaly_Text(vec3<float> (5, 290, 0), vec3<float> (1.0, 1.0, 1.0), GLUT_BITMAP_9_BY_15, "--Pool Game Stats--");`" Thus firstly the position needs to be identified, followed by the colour and then the style, and example of the code used can be seen above.

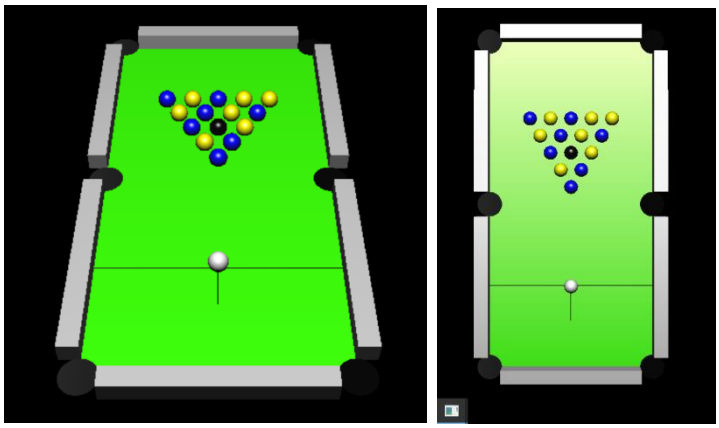

Figure 7: Camera Choice

Camera Option Feature

The last main feature implemented to the pool game was the ability to have two different camera options. This was the simplest feature to implement, as this involved simply adapting already pre-developed features. Firstly, this involved adapting the keyboard feature, so that when a certain key is inputted the camera changes. Secondly within this key input a camera position can then be applied "`gCamPos = vec3<float> (0.0f, 4.0f, 0.01f);`" Thus this allows the user to change the camera position whenever desired.

C++ programming language

C++ is an object orientated programming language based on the C language and was developed in the early 1980s. C++ name comes from its construct "Here "++" use for the extension because "++" is a syntactic construct used in C to increment a variable. Most of C++ content is the super-set of "C", Due to this extension most C programs can be compiled using a C++ compiler" (TekSlate, 2014)

There are a number of different existing programming languages availible to develop many different computer applications, including Java, Python etc. Each language comes with both its advantages and disadvantages, within this section both the advantages and disadvanatages of C++ and Java will be discussed.

C++ Advanatages

There are a number advantages of C++. One of the most promonent advantages of C++ is the ability to use an object orientated programming style (OOP) this is a particular benefit for this instance of a Pool game as it provides for a much more readable design, this is a particular advantage for programs such as a Pool game due to there large size, furthermore in the real world games such as this program are usally updated and improved on a regular basis, thus an object orientated appoach allows for the ability to more easily provide maintance and maintain the application. Furthermore, another significant advantage of C++ in relation to this Pool game is its multi-paradigm programming appoach "paradigm concerned about logics , structure and procedure of the program. C++ is multi-paradigm means it follow three paradigm Generic, Imperative, Object Oriented" (TekSlate, 2014) In terms of the Pool game this is benefical since the game heavily relies on logic based calculations and mathmatic calculation, thus making it an ideal language.

C++ Disadvantages

Although there are a number of advantages of C++, there are also a fair amount of weaknesses. Firstly the language is quite prone to security issues due to pointers and global functions, thus in the real world where game applications need to be secure, it may not be appropriate to develop using a software language prone to flaws “Among other issues, shared variables can cause race conditions in C++ with conditions such as simple increments of variables, loop indices, and shared class objects. Sun Development Network author Phyllis Gustafson advises developers to pay particular” (Turner, 2016)

Java Advantages

In terms of Java, it is also a widely recognised language and has a number of advantages that could be useful for this particular application. Firstly like that of C++ it is Object orientated which allows for an efficient method of developing programs. Secondly probably one of Javas strongest advantages is its interface and does not require a complex knowledge of development “Java has eliminated the use of pointers and also replaced the complexity of multiple inheritances in C++ with a simple structure and that structure is called interface” (Self-Growth, 2016) thus in terms of this application, would more easily allow for people to contribute to the applications development.

Java Disadvantages

Although Java has its clear advantages it again also has its weaknesses. One of the argubaly most significant disadvantages of Java is its memory management “This is one of the big problems in the Java language and dewsvelopers of the language haven’t been able to overcome this problem. Java takes more memory space than the other native programming languages like C and C++.” P-Programming. (2016) Thus arguably making this programming language less efficient.

Conclusion

On an overal basis there is no right or wrong answer to choosing the best programming language. The best language depends entirely on the enviroment it is required for, such as security, memory management, usability, etc. Additionally every language comes with both its set of weaknesses and strengths. In my opinion I believe that C++ is the most suitable language due to its use of OOP and logic based approach while security is an issue I do not believe it is that problematic for this program.

References

1. BCA. (2014) Billards: A Brief History of the Noble Game of Billiards. [Online] Available from: http://bca-pool.com/?page=39 [Last Accessed 07/01/2017]
2. College Foundation. (2017) Billiards and Pool Information. [Online] Available from: https://www1.cfnc.org/Plan/For_A_Career/Career_Cluster_Profile/Cluster_Article.aspx?articleId=b9ja3r4F4ujkhUSOxJxefwXAP3DPAXXAP3DPAX&cId=yJF7dgNzUI6xZl0h6IInegXAP3DPAXXAP3DPAX&sectionId=1 [Last Accessed 07/01/2017]
3. EPA. (2009) World Eight Ball Pool Federation Rules Unabridged Version Issued January 2009. [Online] Available from: http://www.epa.org.uk/wrules.php [Last Accessed 07/01/2017]
4. TekSlate. (2014) Advantages and Disadvantages of C++ language. [Online] Available from: http://tekslate.com/c-explain-advantages-disadvantages/ [Last Accessed 07/01/2017]
5. S, Turner. (2016) Security vulnerabilities of the top ten programming languages: C, Java, C++, Objective-C, C#, PHP, Visual Basic, Python, Perl, and Ruby. [Online] Available from: http://www.aabri.com/LV2013Manuscripts/LV13090.pdf [Last Accessed 07/01/2017]
6. Self-Growth. (2016) Advantages and Disadvantages of Java Application Development. [Online] Available from: http://www.selfgrowth.com/articles/advantages-and-disadvantages-of-java-application-development [Last Accessed 07/01/2017]
7. P-Programming. (2016) 10 DISADVANTAGES OF JAVA PROGRAMMING LANGUAGE. [Online] Available from: http://www.pprogramming.com/2016/06/10-disadvantages-of-java-programming.html [Last Accessed 07/01/2017]