**Feed the Kitty**

**Project Description:**

Project “Feed the Kitty” is based on a game played by 2 to 8 players, components of the game include mice, a bowl, dice/dices and players. Initially the mice is equally divided between the players, if there are extra mice it is placed in the bowl. The dice used in this game consists of 4 symbols that constitute the 6 face of the dice (some of the symbols are repeated). The 4 symbols are:

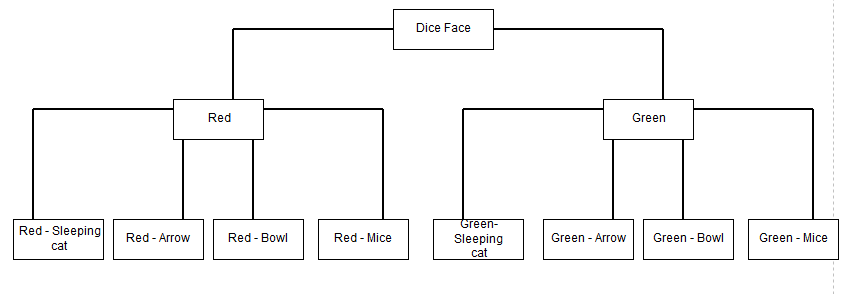
1. Arrow
2. Bowl
3. Mice
4. Sleeping cat

If a player rolls the dice, one of the four symbols appear on top of the dice based on what symbol the player get an action is performed. If the player rolls the dice and gets an arrow, then the player who rolled the dice has to give up one of his/her mice to the next player who is going to roll the dice. If the player rolls the dice and gets a bowl, the player who rolled the dice has to give up one of his mouse to the bowl (Like feeding the cat). Similarly if the player gets a mice which is the luckiest face of the dice a player can get, the player who rolled the dice gets a mice from the bowl. The final symbol left is the sleeping cat if the player a sleeping cat then no action is performed (assumed that the cat is sleeping) and next player gets to roll the dice.

Other than the symbols in the dice there can be color for the dice face as well the dice can contain a face with a color red or a color green, that is if the player gets a mice it might be a green mice or a red mice. The main functionality of the color is to determine what happened to the direction of the game. If green color dice face is acquired then the direction of the game remains same, but if the dice face color is red then the direction of the game will continue in reverse direction until another player get a symbol what is of red color.

In the project the number of players playing the game is determined by a random number that is generated. And also the total number of dice that is used in the game is also based on a random number, even the chance of occurrence of a dice face in a dice is also randomly generated. And the game is an arcade game, this one game is played in one table by 2 to 8 players in one table, similarly there can be more than one table consisting of ‘r’ number of random players with ‘r’ number of random dice and ‘r’ number of random mice.

The project is written in C++ language, the project consists of ‘19’ .cpp files and ‘18’ .h files. The projects uses the following hierarchy to handle the project.



All the blocks listed above are classes used in the project excluding the class Game, arcade, table, player and dice because they do not come under main hierarchy.

The main class “Dice Face” acts as the base class for the child classes Red and Green that are independent of each other, further more these child classes have more child class they are the four symbols used in the game they are the sleeping cat, arrow, bowl and the mice these have separate classes for both the red and the green class as show in the hierarchy block diagram.

The code used in the project is given below.

The Dice class: This is the class where the dice is constructed based on a random number that decides what face should appear on the dice face and what color should the face of the dice should be. This class inherits the functionality of the main\_dice class.

// The Dice class

#ifndef dice

#define dice

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

class m\_dice

{

public:

m\_dice(); // constructor

virtual void display(); // simple display method used to display the generated dice

string\* gendice(); // Dice generate method used to generate a random dice

~m\_dice();

protected:

int f, s, c, \*sa;

};

#endif

Dice cpp code: The following code gives us the definition for all the functions that is used in the header file dice.h, it consists of the definition for the constructor and the destructor and the definition for generating the dice.

#include"dice.h"

using namespace std;

m\_dice::m\_dice()

{

sa = new int[s];

}

void m\_dice::display()

{

int i;

for (i = 0; i < s; i++)

cout << sa[i] << endl;

}

m\_dice::~m\_dice(){delete[]sa; }

string\* m\_dice::gendice()

{

string \*dice1;

dice1 = new string[6];

cout << "was here";

return(dice1);

}

Green color class: As we have seen in the hierarchy diagram the green class should inherit the functionality of the dice face class that is the dice class in this project the header file for the green color class is as follows:

// the green color class

#ifndef color\_g

#define color\_g

#include"dice.h"

using namespace std;

class m\_color\_g :public m\_dice

{

public:

m\_color\_g(); // constructor

virtual void display(); // simple display method

protected:

char color;

};

#endif

Green color cpp code: The following code consists of the definitions used for the functions used in the header file of the color\_g.h header file.

// Definition for green color class

#include"color\_g.h"

using namespace std;

m\_color\_g::m\_color\_g()

{color = 'G';}

void m\_color\_g::display()

{cout << "G";}

Red color class: As we have seen in the hierarchy diagram the red class should inherit the functionality of the dice face class that is the dice class in this project the header file for the red color class is as follows:

// the color red class

#ifndef color\_r

#define color\_r

#include"dice.h"

using namespace std;

class m\_color\_r :public m\_dice

{

public:

m\_color\_r(); // constructor

virtual void display(); // simple display method

protected:

char color;

};

#endif

Red color cpp code: The following code consists of the definitions used for the functions used in the header file of the color\_r.h header file.

#include"color\_r.h"

using namespace std;

m\_color\_r::m\_color\_r() {color = 'R';}

void m\_color\_r::display() { cout << "G"; }

Green arrow class: The following code is the class definition for the green arrow class

// The green arrow class

#ifndef arrow\_g1

#define arrow\_g1

#include"color\_g.h"

using namespace std;

class arrow\_g :public m\_color\_g

{

public:

arrow\_g(); // constructor

string get\_green\_arrow(); // function that returns a green arrow face

virtual void display(); // simple function that displayes a dice face with green arrow

protected:

string symbol;

string sym\_co;

};

#endif

Green arrow cpp: the definition for the member functions of the green\_arrow.h are as follows

// Definition for green arrow class

#include"arrow\_g.h"

using namespace std;

arrow\_g::arrow\_g()

{

symbol = "AR";

}

string arrow\_g::get\_green\_arrow()

{

sym\_co = "G-AR";

return (sym\_co);

}

void arrow\_g::display()

{

m\_color\_g::display();

cout << "AR";

}

Red arrow class: The following code is the class definition for the red arrow class

// The red arrow class

#ifndef arrow\_r1

#define arrow\_r1

#include"color\_r.h"

using namespace std;

class arrow\_r :public m\_color\_r

{

public:

arrow\_r(); // constructor

string get\_red\_arrow(); // member function that return a dice face that is a red arrow

virtual void display(); // fucniton that returns a dice face of red arrow

protected:

string symbol;

string sym\_co;

};

#endif

Red arrow cpp: the definition for the member functions of the red\_arrow.h are as follows

// Definition for red arrow class

#include"arrow\_r.h"

using namespace std;

arrow\_r::arrow\_r()

{

symbol = "AR";

}

string arrow\_r::get\_red\_arrow()

{

sym\_co = "R-AR";

return (sym\_co);

}

void arrow\_r::display()

{

m\_color\_r::display();

cout << "AR";

}

Green Bowl header file: this header file consists of member function to create a dice face of green bowl and display a dice face of a green bowl.

// Green bowl header file

// Class green bowl

#ifndef bowl\_g1

#define bowl\_g1

#include"color\_g.h"

using namespace std;

class bowl\_g :public m\_color\_g

{

public:

bowl\_g(); // constructor

string get\_green\_bw(); // member function to generate a dice face of green bowl

virtual void display(); // funtion to display a dice face of green bowl

protected:

string symbol;

string sym\_co;

};

#endif

Green bowl cpp: the definition for the member functions of the green\_bowl.h are as follows

// memberfunction definitions for green bowl header

#include"bowl\_g.h"

using namespace std;

bowl\_g::bowl\_g()

{

symbol = "BW";

}

string bowl\_g::get\_green\_bw()

{

sym\_co = "G-BW";

return (sym\_co);

}

void bowl\_g::display()

{

m\_color\_g::display();

cout << "BW";

}

Red bowl header file: this header file consists of member function to create a dice face of red bowl and display a dice face of a red bowl.

// Red bowl class

// header file for red bowl class

#ifndef bowl\_r1

#define bowl\_r1

#include"color\_r.h"

using namespace std;

class bowl\_r :public m\_color\_r

{

public:

bowl\_r(); // constructor

string get\_red\_bw(); // member function to get a dice face of red bowl

virtual void display(); // member funcion to display a dice face of a red bowl

protected:

string symbol;

string sym\_co;

};

#endif

Red bowl cpp: the definition for the member functions of the red\_bowl.h are as follows

#include"bowl\_r.h"

using namespace std;

bowl\_r::bowl\_r()

{

symbol = "BW";

}

string bowl\_r::get\_red\_bw()

{

sym\_co = "R-BW";

return (sym\_co);

}

void bowl\_r::display()

{

m\_color\_r::display();

cout << "BW";

}

Green mice header file: this header file consists of member function to create a dice face of green mice l and display a dice face of a green mice.

// green Mice class

#ifndef mice\_g1

#define mice\_g1

#include"color\_g.h"

using namespace std;

class mice\_g :public m\_color\_g

{

public:

mice\_g(); // constructor

string get\_green\_mi(); // member function to get a dice face of green mice

virtual void display(); // member function to display a dice face of a green mice

protected:

string symbol;

string sym\_co;

};

#endif

Green mice cpp: the definition for the member functions of the green\_mice.h are as follows

#include"mice\_g.h"

using namespace std;

mice\_g::mice\_g()

{

symbol = "MI";

}

string mice\_g::get\_green\_mi()

{

sym\_co = "G-MI";

return (sym\_co);

}

void mice\_g::display()

{

m\_color\_g::display();

cout << "MI";

}

Red mice header file: this header file consists of member function to create a dice face of red mice l and display a dice face of a red mice.

// Red mice class

#ifndef mice\_r1

#define mice\_r1

#include"color\_r.h"

using namespace std;

class mice\_r :public m\_color\_r

{

public:

mice\_r(); // constructor

string get\_red\_mi(); // member function to get red mice

virtual void display(); // member function to display a dice face of red mice

protected:

string symbol;

string sym\_co;

};#endif

Red mice cpp: the definition for the member functions of the red\_mice.h are as follows

#include"mice\_r.h"

using namespace std;

mice\_r::mice\_r()

{

symbol = "MI";

}

string mice\_r::get\_red\_mi()

{

sym\_co = "R-MI";

return (sym\_co);

}

void mice\_r::display()

{

m\_color\_r::display();

cout << "MI";

}

Red sleeping cat header file: this header file consists of member function to create a dice face of red sleeping cat l and display a dice face of a red sleeping cat.

// red sleeping cat class header

#ifndef sleeping\_cat\_r1

#define sleeping\_cat\_r1

#include"color\_r.h"

using namespace std;

class sleeping\_cat\_r :public m\_color\_r

{

public:

sleeping\_cat\_r(); // constructor

string get\_red\_sc(); // member function to return red sleeping cat dice face

virtual void display(); // member function to display a red sleeping cat dice face

protected:

string symbol;

string sym\_co;

};#endif

Red sleeping cat cpp: the definition for the member functions of the red\_sleeping cat.h are as follows

#include"sleeping\_cat\_r.h"

using namespace std;

sleeping\_cat\_r::sleeping\_cat\_r()

{

symbol = "SC";

}

string sleeping\_cat\_r::get\_red\_sc()

{

sym\_co = "R-SC";

return (sym\_co);

}

void sleeping\_cat\_r::display()

{

m\_color\_r::display();

cout << "AR";

}

Green sleeping cat header file: this header file consists of member function to create a dice face of green sleeping cat l and display a dice face of a green sleeping cat.

// Green sleeping cat class header

#ifndef sleeping\_cat\_g1

#define sleeping\_cat\_g1

#include"color\_g.h"

using namespace std;

class sleeping\_cat\_g:public m\_color\_g

{

public:

sleeping\_cat\_g(); // constructor

string get\_green\_sc(); // member function to return green sleeping cat dice face

virtual void display(); // member function to display a green sleeping cat dice face

protected:

string symbol;

string sym\_co;

};

#endif

Green sleeping cat cpp: the definition for the member functions of the green\_sleeping\_cat.h are as follows

#include"sleeping\_cat\_g.h"

using namespace std;

sleeping\_cat\_g::sleeping\_cat\_g()

{

symbol = "SC";

}

string sleeping\_cat\_g::get\_green\_sc()

{

sym\_co = "G-SC";

return (sym\_co);

}

void sleeping\_cat\_g::display()

{

m\_color\_g::display();

cout << "SC";

}

Other than the class that is shown in the hierarchy diagram there are few more class used in this project they are

1. Game
2. Table
3. Player
4. Dice
5. Arcade

The header file and the definition for the header file in the cpp file are as follows

Game header file:

// Game header file

#ifndef GAME

#define GAME

#include "main\_dice.h"

#include "Player.h"

#include <fstream>

#include <time.h>

using namespace std;

class m\_game

{

private:

public:

m\_game();

~m\_game();

int set\_players(int);

void set\_dice\_num(int);

void set\_mice\_num(int);

int dice\_roll(string \*dice1);

string\* create\_dice();

void gamedirection(bool);

void play(int players, int dice\_num, int mice\_num,string \*dice1,string \*dice2);

void checkGameStatus(int player,int,int);

void results();

protected:

main\_dice\* dices;

int i,mice\_num, num\_players,dice\_num, mouseforplayer, bowlmouse;

string \*Mice, s;

ifstream fin;

int gamecnt = 0, mflag = 1,l,ln,x1=0,y1=0,oo=2,ll;

bool gamed=true;

string dface1, dface2;

player \*players = new player[num\_players+1];

player currentplayer, nextplayer;};#endif

Game definition file:

#include "game.h"

using namespace std;

m\_game::m\_game()

{

}

m\_game::~m\_game()

{

delete[] dices;

delete[] Mice;

delete[] players;

}

int m\_game::set\_players(int num\_players)

{

cout << "The total number of players are " << num\_players << endl;

for (int i = 0; i < num\_players; i++)

players[i].ID = (i + 1);

return(num\_players);

}

void m\_game::set\_dice\_num(int num\_dices)

{

cout << "The total number of Dice are " << num\_dices << endl;

main\_dice \*dices;

dices = new main\_dice[num\_dices];

}

void m\_game::set\_mice\_num(int mices)

{

cout << "The total number of mice used in game are " << mices << endl;

}

string\* m\_game::create\_dice()

{

int cr, sr, i, s0, s11, s2, s3, t = 1, co, f, c, s1, \*sa, \*ca;

string s;

ifstream in;

in.open("input.txt");

in >> f >> c >> s1;

sa = new int[s1];

ca = new int[c];

string \*symb = new string[s1];

string \*cao = new string[s1];

for (i = 0; i < c; i++)

{

in >> cao[i];

in >> ca[i];

}

for (i = 0; i < s1; i++)

{

in >> symb[i];

in >> sa[i];

}

for (i = 1; i < 4; i++)

sa[i] = sa[i - 1] + sa[i]; string \*dice1; dice1 = new string[6];

s0 = sa[0]; s11 = sa[1]; s2 = sa[2]; s3 = sa[3]; co = ca[0];

arrow\_g ga; bowl\_g gb; mice\_g gm; sleeping\_cat\_g g;

arrow\_r ra; bowl\_r rb; mice\_r rm; sleeping\_cat\_r r;

for (i = 0; i < f; i++)

{

cr = rand() % 99 + 1;

sr = rand() % 99 + 1;

if (sr <= s0)

{

if (cr <= co)

{

s = g.get\_green\_sc();

dice1[i] = s;

}

else

{

s = r.get\_red\_sc();

dice1[i] = s;

}

}

if ((sr > s0) && (sr <= s11))

{

if (cr < co) {

s = ga.get\_green\_arrow();

dice1[i] = s;

}

else {

s = ra.get\_red\_arrow();

dice1[i] = s;

}

}

if ((sr > s11) && (sr <= s2))

{

if (cr < co) {

s = gb.get\_green\_bw();

dice1[i] = s;

}

else {

s = rb.get\_red\_bw();

dice1[i] = s;

}

}

if ((sr > s2) && (sr <= s3))

{

if (cr < co) {

s = gm.get\_green\_mi();

dice1[i] = s;

}

else

{

s = rm.get\_red\_mi();

dice1[i] = s;

}

}

}

return dice1;

//dices[cnt++]=dice1;

}

void m\_game::checkGameStatus(int player, int x1, int y1)

{

cout << "Current player: " << currentplayer.ID << endl;

if (((x1 == 0) && (y1 == 0)) || ((x1 == 1) && (y1 == 1)))

{

cout << "Game Direction: No Change" << endl;

gamed = true;

}

else

{

cout << "Game Direction: Changes" << endl;

gamed = false;

}

if (oo == 1)

gamedirection(gamed);

oo = 1;

cout << "Next player: " << nextplayer.ID << endl;

cout << "-----------------------------------------------------------------------------------------" << endl;

cout << "Bowl" << '\t';

for (i = 0; i < player; i++)

cout<<"Player-"<<players[i].ID << '\t';

cout << endl<<bowlmouse<<'\t';

for (i = 0; i < player; i++)

cout << players[i].mymice << '\t' << '\t';

cout << endl;

cout << "-----------------------------------------------------------------------------------------" <<gamecnt<< endl;

currentplayer.ID = nextplayer.ID;

gamedirection(gamed);

}

void m\_game::gamedirection(bool gamed)

{

if (gamed == true)

{

if (currentplayer.ID == num\_players)

{

nextplayer.ID = 1;

ll = nextplayer.ID;

if (players[ll - 1].mymice == 0)

{

currentplayer.ID = nextplayer.ID;

gamedirection(gamed);

}

}

else

{

nextplayer.ID = currentplayer.ID + 1;

ll = nextplayer.ID;

if (players[ll - 1].mymice == 0)

{

currentplayer.ID = nextplayer.ID;

gamedirection(gamed);

}

}

}

else

{

if (currentplayer.ID == 1)

{

nextplayer.ID = num\_players;

ll = nextplayer.ID;

if (players[ll - 1].mymice == 0)

{

currentplayer.ID = nextplayer.ID;

gamedirection(gamed);

}

}

else

{

nextplayer.ID = currentplayer.ID - 1;

ll = nextplayer.ID;

if (players[ll - 1].mymice == 0)

{

currentplayer.ID = nextplayer.ID;

gamedirection(gamed);

}

}

}

}

void m\_game::play(int player, int dice\_num, int mice\_num,string \*dice1,string \*dice2)

{

try {

num\_players = player;

if (mice\_num > player)

{

mouseforplayer = (mice\_num / player);

for (i = 0; i < player; i++)

players[i].mymice = mouseforplayer;

bowlmouse = mice\_num - mouseforplayer\*player;

}

currentplayer.ID = 1; nextplayer.ID = 2;

checkGameStatus(player, x1, y1);

while (gamecnt<1000)

{

for (i = 0; i < player; i++)

{

if (players[i].mymice == 0)

mflag++;

}

if (mflag == (num\_players))

{

gamecnt = 1000;

goto breakloop;

}

else

{mflag = 1;}

x1 = dice\_roll(dice1);

y1 = dice\_roll(dice2);

checkGameStatus(player, x1, y1);

breakloop:gamecnt++;

}

results();

}

catch (exception ex) {

cout << "";

}

}

int m\_game::dice\_roll(string \*dice1)

{

int randomdiceno;

randomdiceno=rand()% 5+1;

//return(dice1[randomdiceno]);

dface1 = dice1[randomdiceno];

if (dface1 == "G-SC")

{

cout << "Player-" <<currentplayer.ID << " Green - Sleep" << endl;

return (0);

}

if (dface1 == "R-SC")

{

cout << "Player-" << currentplayer.ID << " Red - Sleep" << endl;

return (1);

}

if (dface1 == "G-BW")

{

cout << "Player-" << currentplayer.ID << " Green - Bowl" << endl;

l = currentplayer.ID;

if (players[l-1].mymice>0)

{

players[l - 1].mymice--;

bowlmouse++;

}

return (0);

}

if (dface1 == "R-BW")

{

cout << "Player-" << currentplayer.ID << " Red - Bowl" << endl;

l = currentplayer.ID;

if (players[l - 1].mymice > 0)

{

players[l - 1].mymice--;

bowlmouse++;

}

return (1);

}

if (dface1 == "G-MI")

{

cout << "Player-" << currentplayer.ID << " Green - Mouse" << endl;

l = currentplayer.ID;

if (bowlmouse > 0)

{

players[l - 1].mymice++;

bowlmouse--;

}

return (0);

}

if (dface1 == "R-MI")

{

cout << "Player-" << currentplayer.ID << " Red - Mouse" << endl;

l = currentplayer.ID;

if (bowlmouse > 0)

{

players[l - 1].mymice++;

bowlmouse--;

}

return (1);

}

if (dface1 == "G-AR")

{

cout << "Player-" << currentplayer.ID << " Green - Arrow" << endl;

l = currentplayer.ID;

if (players[l-1].mymice > 0)

{

players[l - 1].mymice--;

ln = nextplayer.ID;

players[ln - 1].mymice++;

}

return (0);

}

if (dface1 == "R-AR")

{

cout << "Player-" << currentplayer.ID << "Red - Arrow" << endl;

l = currentplayer.ID;

if (players[l-1].mymice > 0)

{

players[l - 1].mymice--;

ln = nextplayer.ID;

players[ln - 1].mymice++;

}

return (1);

}

}

void m\_game::results()

{

for (i = 0; i < num\_players; i++)

{

if (players[i].mymice != 0)

{

cout << "Player-" << i + 1 << " WON THE GAME!!!....";

cout << " With totally " << players[i].mymice << " mice" << endl;

}

}

}

Main dice header file:

#ifndef DICE1

#define DICE1

#include"dice.h"

#include"arrow\_g.h"

#include"arrow\_r.h"

#include"bowl\_g.h"

#include"bowl\_r.h"

#include"color\_g.h"

#include"color\_r.h"

#include"mice\_g.h"

#include"mice\_r.h"

#include"sleeping\_cat\_g.h"

#include"sleeping\_cat\_r.h"

#include<time.h>

#include<string>

#include<string.h>

using namespace std;

class main\_dice

{

private:

protected:

public:

main\_dice();

~main\_dice();

void display();

string diceID;

string roll(string dices);

string \*dices=new string[6];

string \*faces = new string[6];

string dice2;

private:

protected:

};#endif

Main dice definition file:

#include"main\_dice.h"

main\_dice::main\_dice(){

}

main\_dice::~main\_dice(){}

void main\_dice::display()

{

for(int i=0;i<6;i++)

cout << dice2[i];

cout <<endl;

}

string main\_dice::roll(string dices)

{

int x;

string s;

x = rand() % 5 + 1;

s = dices[x];

return(s);

}