MTRX3760 Lab 4:

Games Arcade: Dice and Noughts and Crosses

SID: 490476145

Tute: Friday 9am

Functionality:

```
PS
C:\Users\alext\Desktop\Uni\2021\MTRX3760\La
b4\Games Arcade> ./output.exe

Games:
(0) Dice Race Game
(1) Noughts and Crosses
(2) QUIT

What would you like to play?: 0

Game Strategies:
(0) Random
(1) Charge
(2) Smart


What strategy would you like to use?: 1
Your strategy won 50.3% of the time

(Press Enter to Continue)


Games:
(0) Dice Race Game
(1) Noughts and Crosses
(2) QUIT

What would you like to play?: 1

Player Types:
(0) Human Player
(1) Computer Player


Is Player 1 a human or computer?: 0

Is Player 2 a human or computer?: 1

Noughts & Crosses:

+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+

Player 1's turn.

What move would you like to make? (1-9): 1


 Player 1 Moves: 1


+---+---+---+
| X | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+

 Player 2 Moves: 9            ... CONTINUED
```

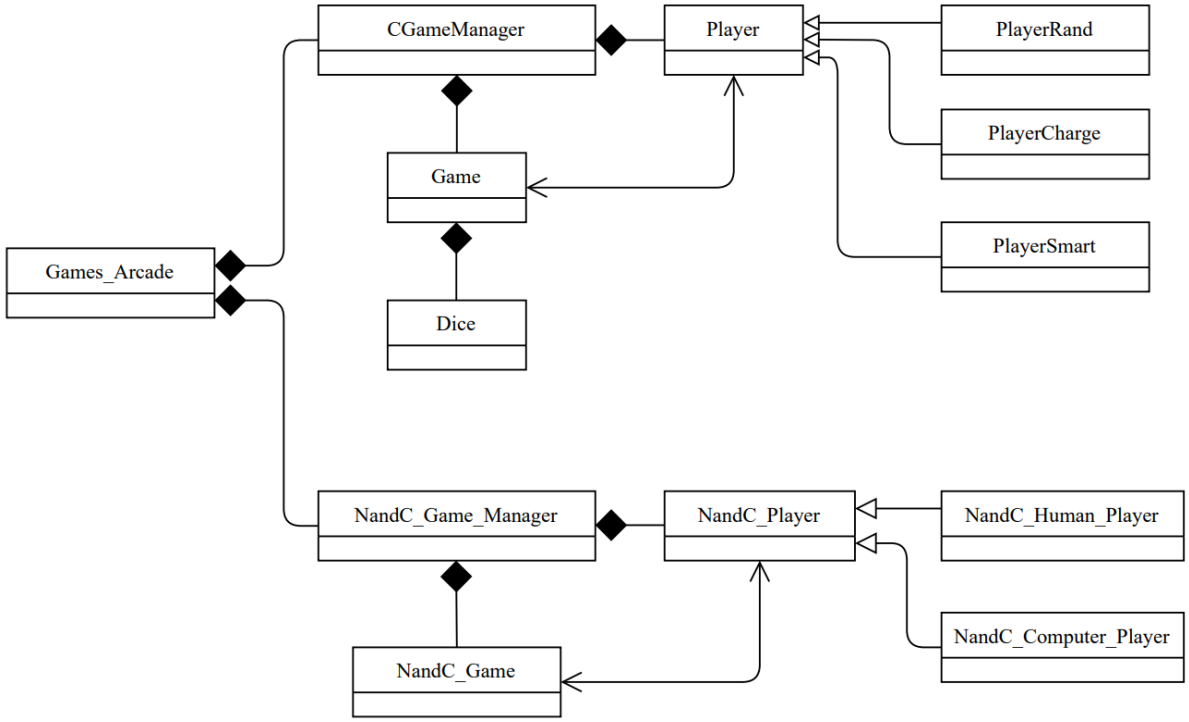
```
...

+---+---+---+
| X | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | O |
+---+---+---+

Player 1's turn.

What move would you like to make? (1-9): 4


 Player 1 Moves: 4


+---+---+---+
| X | 2 | 3 |
+---+---+---+
| X | 5 | 6 |
+---+---+---+
| 7 | 8 | O |
+---+---+---+

 Player 2 Moves: 3


+---+---+---+
| X | 2 | O |
+---+---+---+
| X | 5 | 6 |
+---+---+---+
| 7 | 8 | O |
+---+---+---+

Player 1's turn.

What move would you like to make? (1-9): 7


 Player 1 Moves: 7


+---+---+---+
| X | 2 | O |
+---+---+---+
| X | 5 | 6 |
+---+---+---+
| X | 8 | O |
+---+---+---+

Player 1 wins

(Press Enter to Continue)


Games:
(0) Dice Race Game
(1) Noughts and Crosses
(2) QUIT

What would you like to play?: 2
```

GIT Log:

Output from git log –oneline:

```
…MTRX3760\Lab4\Games Arcade> git log --oneline
1b68bd7 (HEAD -> master, origin/master, origin/HEAD) Polished overall user
experience and commented code
90a874a Finished noughts and crosses and integrated into game arcade
23c6128 More development of nougths and crosses
abde270 Started developing noughts and crosses
e3049a8 Integrating Dice Race into the arcade
2adc4e5 Created Lab 4 Class Structure
66db2da Created Lab 4 Directories
```

UML Diagram:



Above: Games Arcade UML Diagram

Appendix:

Code Written for Lab 1:


```
1  // Enums.h
2  //
3  // This file contains several Enums used throught the project.
4  //
5  // Author: Student 490476145 USYD
6
7
8  #ifndef ENUMS_H
9  #define ENUMS_H
10
11 enum PlayStrategy   // enum defining the types of possible Strategies.
12 {
13     Random,         // Random until the last 12 tiles where player charges.
14     Charge,         // Charge: Always Roll; RollingEnd.
15     Smart
16 };
17
18 enum MoveType       // enum defining the types of possible move.
19 {
20     Roll,           // Roll: roll the dice and advance.
21     Delay           // Delay: delay the opponent by DelayAmount.
22 };
23
24 enum PlayerType     // enum defining the types of possible players.
25 {
26     Human,
27     Computer
28 };
29
30 enum GameType       // enum defining the possible games to play.
31 {
32     DiceRace,
33     NoughtsCrosses
34 };
35
36 #endif
```

```
1  // Games_Arcade.h
2  //
3  // Class used to setup games via a menu system.
4  //
5  // Author: Student 490476145 USYD
6
7  #ifndef _GAMES_ARCADE_H
8  #define _GAMES_ARCADE_H
9
10 #include "Enums.h"
11
12
13 class CGames_Arcade
14 {
15
16     public:
17
18     void menu();                    // Opens the user menu.
19     void WaitForEnter();            // Halts program until player presses enter.
20     void DiceRaceMenu();            // Opens the Dice Race menu.
21     void NoughtsCrossesMenu();      // Opens the Noughts and Crosses menu.
22
23 };
24
25
26 #endif
```

```h
// NandC_Game_Manager.h
//
// Class used to setup noughts and crosses games.
//
// Author: Student 490476145 USYD


#ifndef _NANDC_GAME_MANAGER_H
#define _NANDC_GAME_MANAGER_H

#include "Enums.h"
#include "NandC_Game.h"

class NandC_Game_Manager
{
    public:
        void MakeGame(PlayerType p1, PlayerType p2);  // Creates and starts a game.

    private:
        NandC_Game Game;    // Stores the current game.

};


#endif
```

```
1  // NandC_Game.h
2  //
3  // Class used to run and control noughts and crosses games.
4  //
5  // Author: Student 490476145 USYD
6
7  #ifndef _NANDC_GAME_H
8  #define _NANDC_GAME_H
9
10 #include "Enums.h"
11
12 class NandC_Player;
13
14 class NandC_Game
15 {
16     public:
17         enum Symbol         // enum defining the types of possible symbols.
18         {
19         Blank,              // No Symbol.
20         Cross,              // Cross Symbol.
21         Nought              // Nought Symbol.
22         };
23
24         NandC_Game();                                   // Constructor
25         void Print_Board();                             // Prints the game board.
26         void AddPlayer( NandC_Player* apPlayer );       // Adds a player to the game
27         int Run();                                      // Starts game of N&C
28         bool CheckValid(int Move);                      // Checks move is valid.
29         void Move(int WhoseTurn, int MovePosition);     // Applies players move.
30         bool CheckFinished();                           // Checks if the game is finished.
31
32
33     private:
34         int _Board[3][3] = {0,0,0,0,0,0,0,0,0};         // The game board.
35         NandC_Player* mpPlayers[2];                     // Pointers to the two players.
36         char Print_Symbol(int Symbol,int Coordinate);   // used to print the correct values
   to the game board.
37
38
39 };
40
41
42
43
44 #endif
```

```
1  // NandC_Player.h
2  //
3  // Class used to make player decisions from either user input or computer logic.
4  //
5  // Author: Student 490476145 USYD
6
7
8  #ifndef _NANDC_PLAYER_H
9  #define _NANDC_PLAYER_H
10
11 #include "NandC_Game.h"
12
13
14
15 //-------------------------------------------------------
16 // Player Base Class
17 //
18 class NandC_Player
19 {
20     public:
21
22         NandC_Player( NandC_Game* apTheGame, int aID );     // CTOR: create a player who
   knows the game and has an ID
23         virtual int ChooseMove() = 0;                      // function to choose a move
24         int GetID();                                       // return this player's ID
25
26     protected:
27
28         NandC_Game* mpTheGame;              // keep a pointer to the game (knows-a)
29         int mID;                            // this player's ID, 0 or 1, corresponds to
   their marker ID
30         bool CheckMove(int Move);           // Used to check move is valid.
31
32 };
33
34 //-------------------------------------------------------
35 // Dirived Player Class, Human Player
36 //
37 class NandC_Human_Player: public NandC_Player
38 {
39   public:
40     NandC_Human_Player( NandC_Game* apTheGame, int aID );   // CTOR: create a player who
   knows the game and has an ID
41     int ChooseMove();                                       // Define this player's way of
   choosing moves
42 };
43
44
45 //-------------------------------------------------------
46 // Dirived Player Class, Computer Player
47 //
48 class NandC_Computer_Player: public NandC_Player
49 {
50   public:
51     NandC_Computer_Player( NandC_Game* apTheGame, int aID );    // CTOR: create a player who
   knows the game and has an ID
52     int ChooseMove();                                          // Define this player's way
   of choosing moves
53 };
54
55 #endif
```

```
1  // CDice.h
2  //
3  // Example solution for Lab 2 Polymorphic Dice Race
4  //
5  // Copyright (c) Donald Dansereau, 2021
6
7  #ifndef __CDICE_H
8  #define __CDICE_H
9
10 //-------------------------------------------------------
11 // Simple dice class, knows how to roll a pair of dice
12 class CDice
13 {
14   public:
15     int RollTwoDice();  // rolls two dice and returns the sum
16 };
17
18
19 #endif
```

```
1  // CGame.h
2  //
3  // Example solution for Lab 2 Polymorphic Dice Race
4  //
5  // Copyright (c) Donald Dansereau, 2021
6
7  #ifndef __CGAME_H
8  #define __CGAME_H
9
10
11 //--Includes-----------------------------------------
12 #include "CDice.h"
13
14 //--Forward Declarations---------------------------------
15 class CPlayer;
16
17 //-----------------------------------------------------
18 // Dice race game from Lab 1
19 // If this was well written the first time, it should require
20 // relatively little change for Lab 2
21 // See lab 1 for details and rules
22 class CGame
23 {
24   public:
25     // types
26     enum MoveType                          // enum defining the types of possible move
27     {
28       Roll, Delay                          // roll: roll the dice and advance; delay: delay
the opponent by DelayAmount
29     };
30
31     // consts
32     static const int BoardLength = 64;           // constant defining the length of the race
board
33     static const int DelayAmount = 12;           // if a player chooses to delay their
opponent, by how much are they delayed
34
35     // functions
36     CGame();                                      // Starts without players, resets marker
positions
37     void ResetMarkers();                          // resets markers to starting position
38     void AddPlayer( CPlayer* apPlayer );          // Adds a player to the game; the player
knows their ID
39
40     int GetMarkerPos( int PlayerID );             // returns position of marker for
requested player ID
41
42     bool IsDone();                                // determines if someone's won
43     int WhoWins();                                // returns the winner in case of game
being complete
44
45     void Run();                                   // Runs a complete game
46
47   private:
48     // Mambers
49     int mMarkerPos[2];      // Position of each marker; could also be in a separate board
class
50     CDice mDice;            // This game has dice
51     CPlayer* mpPlayers[2];  // Pointers to the two players; starts empty; you must add
players via AddPlayer()
52
53     // Helper functions
```

```
54      // Move the identified marker by the prescribed amount
55      // invalid requests (off end of board) are ignored
56      void MoveMarker( int WhichMarker, int MoveAmount );
57  };
58
59  #endif
```

```
1  // CGameEvaluator.h
2  //
3  // Example solution for Lab 2 Polymorphic Dice Race
4  //
5  // Copyright (c) Donald Dansereau, 2021
6
7  #ifndef __CGAMEEVALUATOR_H
8  #define __CGAMEEVALUATOR_H
9
10 //--Includes-----------------------------------------------
11 #include "Enums.h"
12 #include "CGame.h"
13
14 //-------------------------------------------------------
15 // Class for evaluating games using different strategies
16 class CGameManager
17 {
18   public:
19     // Knows about and evaluates all alternative player strategies
20     void EvalMethod(PlayStrategy p2);
21
22   private:
23     // Consts
24     static const int mcNReps = 10000; // how many times should the game be run when
   collecting stats
25
26     // Members
27     CGame mGame;              // The game to be evaluated
28
29     // Helper functions
30     double EvalOneMethod();   // evaluates a single method, after mGame is set up with
   appropriate players
31 };
32
33 #endif
```

```
// CPlayer.h
//
// Example solution for Lab 2 Polymorphic Dice Race
//
// This example puts all CPlayers (base class and derived) in one file
// For any more complicated example, you would want to split these out
// into separate files.
//
// Note the init process keeps a pointer to the game, to get info about
// markers. Then during operation no pointers need be passed about, e.g.
// to ChooseMove().
//
// Copyright (c) Donald Dansereau, 2021

#ifndef __CPLAYER_H
#define __CPLAYER_H

//--Includes-----------------------------------------------
#include "CGame.h"

//-------------------------------------------------------
// Abstract base class for a player
class CPlayer
{
  public:

    CPlayer( CGame* apTheGame, int aID );     // CTOR: create a player who knows the game and has an ID
    virtual CGame::MoveType ChooseMove() = 0; // function to choose a move
    int GetID();                              // return this player's ID

  protected:

    CGame* mpTheGame;                         // keep a pointer to the game (knows-a)
    int mID;                                  // this player's ID, 0 or 1, corresponds to their marker ID
};

//-------------------------------------------------------
// Derived class, random player, selects move randomly
class CPlayerRand: public CPlayer
{
  public:
    CPlayerRand( CGame* apTheGame, int aID ); // CTOR: create a player who knows the game and has an ID
    CGame::MoveType ChooseMove();             // Define this player's way of choosing moves
};

//-------------------------------------------------------
// Derived class, charge player, always advances
class CPlayerCharge: public CPlayer
{
  public:
    CPlayerCharge( CGame* apTheGame, int aID ); // CTOR: create a player who knows the game and has an ID
    CGame::MoveType ChooseMove();               // Define this player's way of choosing moves
};

//-------------------------------------------------------
// Derived class, smart player, uses board state to decide
class CPlayerSmart: public CPlayer
```

```
58 {
59   public:
60     CPlayerSmart( CGame* apTheGame, int aID );  // CTOR: create a player who knows the game
   and has an ID
61     CGame::MoveType ChooseMove();               // Define this player's way of choosing
   moves
62 };
63
64 class CPlayerHuman: public CPlayer
65 {
66   public:
67     CPlayerHuman( CGame* apTheGame, int aID ); // CTOR: create a player who knows the game
   and has an ID
68     CGame::MoveType ChooseMove();               // Define this player's way of choosing
   moves
69 };
70
71 #endif
```

```cpp
// Main.cpp
//
// Author: Student 490476145 USYD

#include <stdlib.h>

#include "Games_Arcade.h"

int main()
{
    srand(4322);

    // Creates Game Arcade and opens the menu.
    CGames_Arcade gamesArcade = CGames_Arcade();
    gamesArcade.menu();

    return 0;
}
```

```cpp
#include "Enums.h"
#include "NandC_Game_Manager.h"
#include "NandC_Player.h"

//-------------------------------------------------------
void NandC_Game_Manager::MakeGame(PlayerType p1, PlayerType p2)
{
    // Defines Types of possible players.
    NandC_Computer_Player computer1(&Game,0);
    NandC_Computer_Player computer2(&Game,1);
    NandC_Human_Player human1(&Game,0);
    NandC_Human_Player human2(&Game,1);

    // Add Player 1 to Game.
    switch (p1)
    {
    case Human:
        Game.AddPlayer(&human1);
        break;

    case Computer:
        Game.AddPlayer(&computer1);
        break;
    }

    // Add Player 1 to Game.
    switch (p2)
    {
    case Human:
        Game.AddPlayer(&human2);
        break;

    case Computer:
        Game.AddPlayer(&computer2);
        break;
    }

    // Starts Game.
    Game.Run();

}
```

```cpp

#include "NandC_Game.h"
#include "NandC_Player.h"

#include <iostream>

//---------------------------------------------------------
NandC_Game::NandC_Game()
{

    // Resets Game Players.
    for( int iPlayer = 0; iPlayer < 2; ++iPlayer )
    {
        mpPlayers[iPlayer] = NULL;
    }

}

//---------------------------------------------------------
int NandC_Game::Run()
{

    int WhoseTurn = 0;

    std::cout   << "\nNoughts & Crosses:";
    Print_Board();

    while (true)
    {

        // Gets Players Move.
        int MovePosition = mpPlayers[WhoseTurn] -> ChooseMove();

        std::cout << "\n Player " << (WhoseTurn+1) << " Moves: " << MovePosition << "\n";

        // Applies Move.
        Move(WhoseTurn,MovePosition);

        // Displays Game Board.
        Print_Board();

        // Breaks if the game is finished.
        if(CheckFinished())
        {
            std::cout << "\nPlayer " << (WhoseTurn+1) << " wins\n";
            break;
        }

        // Swaps Turn.
        WhoseTurn = (WhoseTurn + 1) % 2;
    }

    return WhoseTurn;
}

//---------------------------------------------------------
void NandC_Game::Move(int WhoseTurn, int MovePosition)
{
    // Converts int Move to x and y coordinates.
    int x = (MovePosition-1) % 3;
    int y = (MovePosition-1) / 3;
```

```
62
63      // Places nought or cross.
64      _Board[x][y] = WhoseTurn + 1;
65  }
66
67  //---------------------------------------------------------
68  void NandC_Game::Print_Board()
69  {
70      int x;
71      int y;
72
73      std::cout   << "\n\n+---+---+---+\n";
74
75      for(y = 0; y<3; y++)
76      {
77         std::cout << "|";
78         for(x = 0; x<3; x++)
79          {
80              std::cout   << " "
81                          << Print_Symbol(_Board[x][y],(x+1)+(y+1)*3)
82                          << " |";
83          }
84          std::cout << "\n+---+---+---+\n";
85      }
86
87  }
88
89  //---------------------------------------------------------
90  char NandC_Game::Print_Symbol(int Symbol,int Coordinate)
91  {
92      char value;
93
94      switch (Symbol)
95      {
96      case Blank:
97          value = Coordinate + 45;
98          break;
99
100     case Cross:
101         value = 'X';
102         break;
103
104     case Nought:
105         value = 'O';
106         break;
107
108     default:
109         break;
110     }
111
112     // Returns X, O or number(0-9), depending on the board value.
113     return value;
114 }
115
116
117 //---------------------------------------------------------
118 void NandC_Game::AddPlayer( NandC_Player* apPlayer )
119 {
120   mpPlayers[apPlayer->GetID()] = apPlayer;
121 }
122
```

```
123 bool NandC_Game::CheckValid(int Move)
124 {
125     // Converts from int Move to x and y coordinates.
126     int X = (Move-1) % 3;
127     int Y = (Move-1) / 3;
128 
129     // Returns true if location is empty.
130     return !_Board[X][Y];
131 }
132 
133 bool NandC_Game::CheckFinished()
134 {
135 
136     int X;
137     int Y;
138 
139     bool Result = false;
140 
141     // Check Vertical.
142     for (X = 0; X<3; X++)
143     {
144         if( (( _Board[X][0] == _Board[X][1] ) && ( _Board[X][0] == _Board[X][2] ))  && (
    _Board[X][0] ) )
145         {
146             Result = true;
147         }
148     }
149 
150     // Check Horisontal.
151     for (Y = 0; Y<3; Y++)
152     {
153         if( (( _Board[0][Y] == _Board[1][Y] ) && ( _Board[0][Y] == _Board[2][Y] )) && (
    _Board[0][Y] ) )
154         {
155             Result = true;
156         }
157     }
158 
159     // Check Diagonal.
160     if ( ((( _Board[1][1] == _Board[0][0] ) && ( _Board[1][1] == _Board[2][2] )) || ((
    _Board[1][1] == _Board[2][0] ) && ( _Board[1][1] == _Board[0][2] ))) && (( _Board[1][1] ))
    )
161 
162     {
163         Result = true;
164     }
165 
166     // Returns true if game has finished.
167     return Result;
168 
169 }
```

```cpp
#include "NandC_Player.h"

#include <cstdlib>      // rand
#include <iostream>
#include <string>

//---------------------------------------------------------
NandC_Player::NandC_Player( NandC_Game* apTheGame, int aID )
  : mpTheGame( apTheGame ), mID( aID )
{
}

//---------------------------------------------------------
int NandC_Player::GetID()
{
  return mID;
}

//---------------------------------------------------------
bool NandC_Player::CheckMove(int Move)
{
  bool Result = false;

  if(mpTheGame -> CheckValid(Move))
  {
    Result = true;
  }
  else
  {
    //std::cout << "Invalid Move.\n";
  }

  return Result;
}


//---------------------------------------------------------
NandC_Human_Player::NandC_Human_Player( NandC_Game* apTheGame, int aID )
  : NandC_Player( apTheGame, aID )
{
}


//---------------------------------------------------------
int NandC_Human_Player::ChooseMove(  )
{

  int Move;

  while (true)
  {
    std::cout << "\nPlayer " << (mID+1) << "'s turn.\n"
              << "\nWhat move would you like to make? (1-9): ";
    std::cin >> Move;
    std::cout << "\n";

    if(CheckMove(Move))
    {
      break;
    }

```

```
62    }
63    return Move;
64  }
65
66
67  //---------------------------------------------------------
68  NandC_Computer_Player::NandC_Computer_Player( NandC_Game* apTheGame, int aID )
69    : NandC_Player( apTheGame, aID )
70  {
71  }
72
73  //---------------------------------------------------------
74  int NandC_Computer_Player::ChooseMove(  )
75  {
76    int Move = 1;
77
78    while (true)
79    {
80
81      Move = (rand() % 9) + 1;
82
83      if(CheckMove(Move))
84      {
85        break;
86      }
87
88    }
89
90    return Move;
91  }
```

```cpp
// CDice.cpp
//
// Example solution for Lab 2 Polymorphic Dice Race
//
// Copyright (c) Donald Dansereau, 2021

//--Includes-----------------------------------------------
#include "CDice.h"

#include <cstdlib>      // rand

//---------------------------------------------------------
// It's important to add two dice, rand() % 12 doesn't do it
int CDice::RollTwoDice()
{
  int Result1 = (rand() % 6) + 1;
  int Result2 = (rand() % 6) + 1;
  int Result = Result1 + Result2;
  return Result;
}
```

```cpp
// CGame.cpp
//
// Example solution for Lab 2 Polymorphic Dice Race
//
// Copyright (c) Donald Dansereau, 2021

#include "CGame.h"
#include "CPlayer.h"

#include <cstddef>

//----------------------------------------------------------
CGame::CGame()
{
  for( int iPlayer = 0; iPlayer < 2; ++iPlayer )
  {
    mpPlayers[iPlayer] = NULL;
  }
  ResetMarkers();
}

//----------------------------------------------------------
void CGame::ResetMarkers()
{
  for( int iPlayer = 0; iPlayer < 2; ++iPlayer )
  {
    mMarkerPos[iPlayer] = 0;
  }
}

//----------------------------------------------------------
void CGame::AddPlayer( CPlayer* apPlayer )
{
  mpPlayers[apPlayer->GetID()] = apPlayer;
}

//----------------------------------------------------------
int CGame::GetMarkerPos( int PlayerID )
{
  return mMarkerPos[PlayerID];
}

//----------------------------------------------------------
void CGame::Run()
{
  int WhoseTurn = 0;
  int IterCount = 0;

  while( true )
  {
    //std::cout << "It is Player " << WhoseTurn << "'s turn: ";
    // better to not have commented out code; we'll learn later how to build a logging class to avoid this
    MoveType WhichMove = mpPlayers[WhoseTurn]->ChooseMove();
    switch( WhichMove )
    {
      case Roll:
      {
        int RollAmount = mDice.RollTwoDice();
        MoveMarker(WhoseTurn, RollAmount);
        //std::cout << "Rolled " << RollAmount << std::endl;
```

```
61        }
62        break;
63        case Delay:
64        {
65          int OtherPlayer = (WhoseTurn+1)%2;
66          MoveMarker( OtherPlayer, -DelayAmount );
67          //std::cout << "Delayed" << std::endl;
68        }
69        break;
70      };
71      if( IsDone() )
72      {
73        //std::cout << "Player " << WhoseTurn << " Wins!" << std::endl;
74        for( int iMarker = 0; iMarker < 2; ++iMarker )
75        {
76          //std::cout << "Marker " << iMarker << " is at " << mMarkerPos[iMarker] <<
std::endl;
77        }
78        break;
79      }
80      WhoseTurn = (WhoseTurn + 1) % 2;
81      ++IterCount;
82      //std::cout << "Iter: " << IterCount << std::endl;
83    }
84  }
85
86  //-------------------------------------------------------
87  bool CGame::IsDone()
88  {
89    bool Result = false;
90    for( int iMarker = 0; iMarker < 2; ++iMarker )
91    {
92      //std::cout << "Marker " << iMarker << " is at " << mMarkerPos[iMarker] << std::endl;
93      if( mMarkerPos[iMarker] == BoardLength )
94      {
95        Result = true;
96      }
97    }
98
99    return Result;
100 }
101
102 //-------------------------------------------------------
103 void CGame::MoveMarker( int WhichMarker, int MoveAmount )
104 {
105   if( (mMarkerPos[WhichMarker] + MoveAmount <= BoardLength) && (mMarkerPos[WhichMarker] +
MoveAmount >= 0) )
106   {
107     mMarkerPos[WhichMarker] += MoveAmount;
108   }
109 }
110
111 //-------------------------------------------------------
112 int CGame::WhoWins()
113 {
114   int Result = -1;
115   for( int iMarker=0; iMarker<2; ++iMarker )
116     if( mMarkerPos[iMarker] == BoardLength )
117       Result = iMarker;
118   return Result;
119 }
```

```cpp
// CGameEvaluator.cpp
//
// Example solution for Lab 2 Polymorphic Dice Race
//
// Copyright (c) Donald Dansereau, 2021

//--Includes-----------------------------------------
#include "CGameManager.h"

#include "CPlayer.h"
#include "CGame.h"
#include "Enums.h"

#include <iostream>

//-----------------------------------------------------
// This evaluator creates players, adds them to the game,
// and runs the game to evaluate them via the helper function
// EvalOneMethod. There are better ways to do this, we will
// study the factory design pattern later on...
// For now a key point is that there are no if statements deciding
// how the player should behave... it's all done through polymorphism.
void CGameManager::EvalMethod(PlayStrategy p2)
{

    CPlayerRand player1( &mGame, 0 );
    CPlayerRand player2( &mGame, 1 );
    CPlayerCharge player3( &mGame, 1 );
    CPlayerSmart player4( &mGame, 1 );

    mGame.AddPlayer( &player1 );

    switch (p2)
    {
    case Random:

        mGame.AddPlayer( &player2 );

        break;

    case Charge:

        mGame.AddPlayer( &player3 );

        break;

    case Smart:

        mGame.AddPlayer( &player4 );

        break;

    default:
        break;
    }




    {   // note the use of curly braces to set up a scope here for myChargePlayer and StrategyWins variables
```

```
61          double StrategyWins = EvalOneMethod();
62          std::cout << "Your strategy won " << StrategyWins << "\% of the time" << std::endl;
63      }
64
65  }
66
67  //--------------------------------------------------------
68  double CGameManager::EvalOneMethod()
69  {
70      double Result = -1;
71      int TrackWins = 0;
72
73      for( int iReps = 0; iReps < mcNReps; ++iReps )
74      {
75          mGame.ResetMarkers();
76          mGame.Run();
77          int Winner = mGame.WhoWins();
78          if( Winner == 1 )
79              ++TrackWins;
80      }
81
82      Result = TrackWins * 100.0 / double(mcNReps);
83      return Result;
84  }
```

```cpp
// CPlayer.cpp
//
// Example solution for Lab 2 Polymorphic Dice Race
//
// Copyright (c) Donald Dansereau, 2021

//--Includes-----------------------------------------------
#include "CPlayer.h"

#include <cstdlib>      // rand
#include <iostream>
#include <string>

//-------------------------------------------------------
CPlayer::CPlayer( CGame* apTheGame, int aID )
  : mpTheGame( apTheGame ), mID( aID )
{
}

//-------------------------------------------------------
int CPlayer::GetID()
{
  return mID;
}

//-------------------------------------------------------
CPlayerRand::CPlayerRand( CGame* apTheGame, int aID )
  : CPlayer( apTheGame, aID )
{
}

//-------------------------------------------------------
CGame::MoveType CPlayerRand::ChooseMove(  )
{
  CGame::MoveType Result;
  if( rand() % 2 == 0 )
  {
    Result = CGame::Roll;
  }
  else
  {
    Result = CGame::Delay;
  }
  return Result;
}

//-------------------------------------------------------
CPlayerCharge::CPlayerCharge( CGame* apTheGame, int aID )
  : CPlayer( apTheGame, aID )
{
}

//-------------------------------------------------------
CGame::MoveType CPlayerCharge::ChooseMove()
{
  CGame::MoveType Result = CGame::Roll;
  return Result;
}

//-------------------------------------------------------
CPlayerSmart::CPlayerSmart( CGame* apTheGame, int aID )
```

```
62    : CPlayer( apTheGame, aID )
63  {
64  }
65
66  //---------------------------------------------------------
67  CGame::MoveType CPlayerSmart::ChooseMove()
68  {
69    CGame::MoveType Result;
70    Result = CGame::Roll;
71
72    int OpponentID = (mID + 1) % 2;
73
74    int OpponentPos = mpTheGame->GetMarkerPos( OpponentID );
75
76    if( abs((CGame::BoardLength - OpponentPos)) < 13 )
77       Result = CGame::Delay;
78
79    return Result;
80  }
81
82  //---------------------------------------------------------
83  CPlayerHuman::CPlayerHuman( CGame* apTheGame, int aID )
84    : CPlayer( apTheGame, aID )
85  {
86  }
87
88  //---------------------------------------------------------
89  CGame::MoveType CPlayerHuman::ChooseMove(  )
90  {
91    CGame::MoveType Result;
92
93    int userInput;
94
95    while (true)
96    {
97      std::cout << "\nWhat move would you like to make? Roll (1) or Delay (2): ";
98      std::cin >> userInput;
99      std::cout << "\n";
100
101     if (userInput == 1)
102     {
103
104       Result = CGame::Roll;
105       break;
106
107     }
108     else if (userInput == 2)
109     {
110
111       Result = CGame::Delay;
112       break;
113
114     }
115
116
117   }
118   return Result;
119 }
```

```cpp
// Games_Arcade.cpp
//
// Methods used to setup games via a menu system.
//
// Author: Student 490476145 USYD


#include <iostream>
#include <string>

#include "Games_Arcade.h"
#include "CGameManager.h"
#include "NandC_Game_Manager.h"


//-------------------------------------------------------
void CGames_Arcade::menu()
{

    int GameSelect;

    while (true)
    {

        std::cout   << "\nGames:\n"
                    << "(0) Dice Race Game\n"
                    << "(1) Noughts and Crosses\n"
                    << "(2) QUIT\n\n"
                    << "What would you like to play?: ";

        std::cin >> GameSelect;

        if(GameSelect == 2){
            break;
        }

        switch (GameSelect)
        {
        case 0:
            DiceRaceMenu();
            break;

        case 1:
            NoughtsCrossesMenu();
            break;

        default:
            std::cout << "\nInvalid input. Try Again: ";
            break;
        }

        WaitForEnter();
    }

}




//-------------------------------------------------------
void CGames_Arcade::DiceRaceMenu()
```

```
62 {
63
64     CGameManager DiceGameManager;
65     int GameStrategy;
66
67     while (true)
68     {
69
70         std::cout   << "\nGame Strategies:\n"
71                     << "(0) Random\n"
72                     << "(1) Charge\n"
73                     << "(2) Smart\n\n";
74
75         std::cout   << "\nWhat strategy would you like to use?: ";
76
77         std::cin >> GameStrategy;
78
79
80         if(GameStrategy<=2)
81         {
82
83             // Starts Dice Race with user defined parameters.
84             DiceGameManager.EvalMethod((PlayStrategy) GameStrategy);
85             break;
86
87         }
88
89
90         std::cout << "\nInvalid input. Try Again: ";
91
92
93     }
94 }
95
96 //-------------------------------------------------------
97 void CGames_Arcade::NoughtsCrossesMenu()
98 {
99
100     int Player1;
101     int Player2;
102
103     while (true)
104     {
105
106         std::cout   << "\nPlayer Types:\n"
107                     << "(0) Human Player\n"
108                     << "(1) Computer Player\n\n";
109
110         std::cout   << "\nIs Player 1 a human or computer?: ";
111
112         std::cin >> Player1;
113
114         std::cout   << "\nIs Player 2 a human or computer?: ";
115
116         std::cin >> Player2;
117
118         // Starts Noughts and Crosses with user defined parameters.
119         NandC_Game_Manager NandC_Manager;
120         NandC_Manager.MakeGame((PlayerType) Player1,(PlayerType) Player2);
121
122         break;
```

```
123     }
124
125 }
126
127 //--------------------------------------------------------
128 void CGames_Arcade::WaitForEnter()
129 {
130     std::cout   << "\n(Press Enter to Continue)\n";
131     std::cin.ignore();
132     std::cin.ignore();
133 }
```