Софийски университет “Св. Климент Охридски”
Факултет по математика и информатика

Дисциплина:	*Обектно ориентирано програмиране*
Преподавател:	*инф. Димитров*
Проект:	*Игра Калах*
Студент:	*Анастасия Радева*
Специалност:	*Информатика*
	фак. № 42441
	II курс, 3-та група

16 Януари 2000г.
Гр. София

1. Описание на проекта

Играта Калах е древна интелектуална игра, известна още и като игра на Маите. Целта на настоящия проект е създаването на компютърна програма за тази игра.

1.1. Формулиране на задачата

Създаване на програма за играта Калах, позволяваща възможността да играят двама играчи, играч срещу компютър и компютър срещу компютър. Да се предвиди възможност за задаване на следните параметри: броя камъчета за всяка лунка и играча, правещ първия ход. Да се осигури избор на компютърната стратегия на игра от тип Random и Min-Max. За Минимаксната процедура да се предвиди задаване дълбочината на анализа, случаен избор при равностойни ходове, запис и разглеждане на информация за проведения анализ.

1.2. Общо описание на играта

Играта се играе от двама играчи. Игралното поле се състои от 14 ямки, по седем за всеки играч, разположени в правоъгълен участък. Ямките са два вида: калах (по един за играч) и лунки (по 6 за всеки). Разположението на ямките, в посока обратна на часовниковата стрелка, е: лунките на първия играч, калаха на първия играч, лунките на втория играч, калаха на втория играч.

Основен атрибут на играта са камъчетата, чийто брой е от 3 до 6 за всяка лунка (или от 36 до 72 общо). Преди започване на играта камъчетата се разпределят по равно в лунките на двамата играчи.

Играта представлява серия от ходове с редуване на играчите. Кой да започне първи е въпрос на споразумение между тях.

Ходовете, подчинени на правилата описани по-долу, са последователност от премествания на камъчета в ямките. Всеки пореден ход се извършва от играча, който е на ред.

Целта на всеки играч е да събере повече от половината камъчета в своя калах, при което играта приключва в негова полза.

1.3. Правила на играта

Ходът представлява избор на една от непразните лунки, принадлежащи на играча на ход. Камъчетата се изваждат от лунката и се поставят последователно по едно в ямките по посока обратна на

часовниковата стрелка. При това поставяне се пропуска само противниковият калах.

При попадане на последното камъче в собствена непразна лунка, при условие, че играчът е поставил камъчета и в противниковите лунки, ходът се продължава от същия играч и от лунката, в която последно е поставено камъче.

При условие, че ходът завърши в противникова лунка, имаща след поставянето на последното камъче 2 или 3 камъчета, те се пленяват и поставят в калаха на играещия хода. Пленяването на камъчета от противниковите лунки продължава в посока на часовниковата стрелка при условие, че броят на камъчетата в тях е 2 или 3.

След завършване на поредния ход играчът проверява броя на камъчетата в своя калах и ако той е повече от половината от общия им брой, играта приключва в негова полза.

Играта може да приключи в полза на играча, изиграл ход и при условие, че противникът му е само с празни лунки. При тази ситуация играчът прибира всички камъчета от лунките си в своя калах и приключва играта с победа.

2. Описание на програмата

Програмата за играта Калах е написана на C++ и предназначена за работа в средата на Windows 95/98.

Средата, използвана за разработката на програмата, е C++ Builder 3 на Borland International.

Програмата се състои от основен прозорец носещ, всички визуални обекти. Обектите, използвани в програмата, са контроли от три типа:

- Контроли за задаване условията на играта;
- Контроли за управление на играта;
- Контроли за визуализиране ходовете на играта.

Програмата включва следните възможности:

- Избор на броя камъчета за лунка – 3-6;
- Избор на първия играч – A или B;
- Избор на типа за всеки играч – Player или Computer;
- Избор на режим за игра на компютъра – Random или Min-Max;

- Избор на параметрите на режима Min-Max:
  - Дълбочина на анализираното дърво – 4-6;
  - Случаен избор при еднакво добри възможни ходове;
  - Визуализация дървото на анализираните ходове.

Общи възможности на програмата:

- Игра в режими:
  - Player – Player;
  - Player – Computer;
  - Computer – Computer.
- Визуализация на всички изиграни ходове.
- Визуализация на двете дървета на анализ при два последователни хода в режим Computer – Computer

2.1. Описание на режимите и обектите

Основните режими на играта са избор на параметрите и игра.

2.1.1. Избор на параметрите на играта

В този режим чрез визуалните контроли в основния прозорец се задават всички параметри на играта изброени в т. 2. Използваните контроли са от библиотеката на визуалната среда за програмиране и могат да се видят в приложението “Main.h”.

Основен обект в играта е GameField – игралното поле, представляващ основната функционална единица на играта. Всички зададени параметри се предават при промяна чрез съответните Event Handlers на контролите на публичните променливи на GameField.

Указатели към визуализиращите контроли от класовете TListView и TTreeView се предават на GameField за целите на визуализацията.

2.1.2. Режим на игра

Управлението на играта (Start, Stop, Step) се контролира от три обекта от тип TButton, активиращи подходящи функции в обекта GameField. По-специално е управлението на играта, в която участва Computer Player в режим на визуализация на дървото на избор на ход. В този случай активирането на ход от страна на Computer Player се осъществява чрез бутона Step във взаимодействие с GameField.

При стартиране на игра цялостният контрол се осъществава от GameField.

2.2. *Описание на специалните класовете*

За реализиране на основните функции на играта са дефинирани два спициални класа TGameField и TLunka, свойствата и функциите на които се виждат в приложенията “GameField.h” и “Lunka.h”.

2.2.1. Клас TLunka

Класът е предназначен за реализиране на функциите на основните игрални обекти на играта – ямките. Класът е наследник на свойствата и функциите на класа TLabel с добавени специални такива.

Три са основните дейности, заложени в този клас.

Поддръжка и визуализация на основните свойства на ямката:

- Тип – калах или лунка;
- Принадлежност – на играч A или B;
- Пореден номер в игралното поле;
- Брой камъчета.

Визуализирането на информацията е в текстов вид и се базира на визуалните възможности на базовия клас. Формата на извежданата информация е “A3-12”, означаваща трета лунка на играча A, съдържаща 12 камъчета.

Обслужва единствено събитието “Click” от мишката. В обекта не се предизвиква вътрешно действие, а единствено при, свързан със свойството OnClick, Event Handler му се предава управлението.

Два указателя Next и Previous (от тип * TLunka) се насочват към съседните й ямки по време на конструирането на GameField, при което се образува двусвързан списък от обекти (ямки). Така получената структура от обекти от тип TLunka чрез три фукции членове на класа (съдържащи правилата на играта), реализира игровия ход на базата на взаимодействие между отделните обекти в структурата. Тези функции са: GoFirst, GoNext, GoPrevious.

Активирането на ход се извършва чрез извикване на GoFirst функцията на избраната лунка. Тя нулира броя на наличните камъчета и извиква GoNext на следващата ямка. Всяка следваща ямка прибавя (ако не е противников калах) по едно камъче към своите и извиква GoNext на следващата ямка с намаления брой

камъчета (ако те не са станали 0). Ако предаваните по веригата на ямките камъчета са привършили, ямката, установила това, проверява условията за приключване на ход, повторен ход или пленяване. При приключване на ход ямката завършва изпълнението на извиканата функция. При условие за повторен ход лунката извършва действията на собствената си GoFirst функция и извиква GoNext на следващата. При възможност за пленяване противниковата лунка нулира броя на камъчетата си и извиква GoPrevious на предходната ямка. В обратен на хода ред пленените камъчета се предават (и при условие за вторично пленяване се увеличават) до първия срещнат калах (принадлежащ на играча на ход),. който от своя страна ги добавя към своите и приключва хода.

При реализирането на функциите GoNext и GoPrevious са спазени изискванията за създаване на рекурсивни функции, поради факта, че последователното им извикване може да доведе до нееднократно извикване на GoNext на един и същи обект в рамките на един ход. Освен това завършването на всеки ход наподобява връщането от рекурсия.

2.2.2. Клас TGameField

Този клас е основен в програмата и фактически реализира цялостната игра. Той е наследник на класа TPanel и представлява визуален контейнер за множеството обекти от тип TLunka, както и всички допълнителни обекти, свойства и функции за управление на играта. Свойствата, функциите и вътрешните обекти на TGameField могат да се класифицират по ролята им:

2.2.2.1. Визуални компоненти на TGameField

Основните визуални компоненти са 14 обекта от тип TLunka, които в конструктора си TGameField създава, свързва в структура (по горе описания начин), предава необходимите стойности на свойствата им и пресъединява Event Handler към OnClick свойството на лунките, принадлежащи на Player. Указателите към създадените обекти се присвояват на вътрешни масиви LunkyA и LunkyB за пряк, индексен достъп. За подобряване на графичния вид на игралното поле се използват и 14 помощни обекта от тип TShape, оформящи лунките като елепси, а калахите като правоъгълници със заоблени върхове.

За осигуряване на визуалната поддръжка при промяна на размера на обекта по време на работа на програмата, в TGameField се обработва съобщението на Windows – WM_SIZE от функцията

WMSize, която извършва промяна на разположението и размерите на принадлежащите му визуални компоненти.

2.2.2.2. Функционални компоненти на TGameField

При началното инициализиране на играта важна функция е ReNumLunky, която предава необходимите стойности на обектите от тип TLunka и пресвързва LunkyClick в зависимост от режима на игра.

При игра Player – Player функционирането се осъществява изцяло от LunkyClick, в която се контролира смяната на играча, активира се GoFirst функцията на избраната за игра лунка, отразява се направения ход в ListView и се следи за края на играта.

При игра Player – Computer Player активирането на ход от страна на Player се извършва чрез мишката и се обработва изцяло от LunkyClick. Активирането на компютърен ход се извършва посредством обект от тип TTimer, който се запуска в LunkyClick след приключване хода на Player. При изтичане на интервала на Timer обекта (време необходимо за завършване изпълнението на LunkyClick) се извиква съответният Event Handler. Той от своя страна извиква функцията за избор на ход (MoveChoice), след което активира LunkyClick от името на избраната лунка.

При игра Computer Player – Computer Player след стартирането на играта се запуска таймерът на първия играч. Обслужването на хода става както е описано по-горе с тази разлика, че LunkyClick алтернативно активира таймера на следващия играч.

2.2.2.3. Интелектуални компоненти на TGameField

Две са възможностите за избор на ход от Computer Player– Random и Min-Max. Те се реализират чрез по-горе споменатата функция MoveChoice. Специфичен е случаят, когато Computer Player не прави случаен избор на ход, а анализира текущатата ситуация и избира възможно най-добрия ход, осигуряващ му победа или преднина. За тази цел в програмата е вложена Минимаксната процедура (описана подробно в литературата), която анализира всички възможни ходове и на двамата играчи до предварително зададената дълбочина на анализа. Тя е реализирана чрез рекурсивната функция Rfunc. За проиграването на ходовете при анализа се използва структурата от обекти от тип TLunka, като преди всяко стартиране на ход се съхранява състоянието на обектите, което се възстановява след извършването му. На всяко ниво на рекурсията в права посока се записва направеният ход, а в обратната посока на даденото ниво се добавя стойността на оценачната функция,

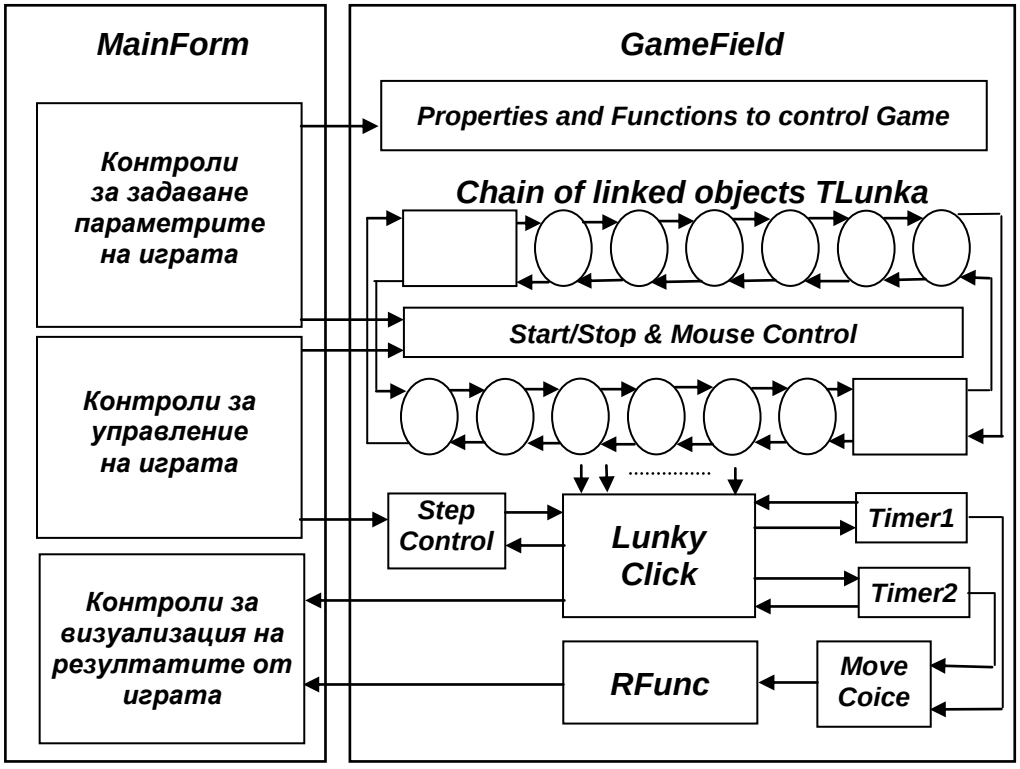
необходима за Min-Max. Тази информация се записва в обект от тип TStrings в подходящ за дървовидно представяне формат. След завършването на процедурата тази информация се прехвърля чрез файл в съответния обект TreeView за разглеждане.

Рекурсивната процедура в зависимост от дълбочината на анализа може да доведе до изчерпване на стековата и даннова памет, заделена за програмата. Поради тази причина се налага нейното оптимизиране по отношение на броя и размера на формалните параметри и локалните променливи, както и ограничаване на максимално допустимата дълбочина на рекурсията. Използваният обект от тип Tstrings за запис на анализираната информация се осигурява динамично с памет от общия пул на системата, което не нарушава баланса на паметта на програмата.

3. Програмната структура и интерфейс

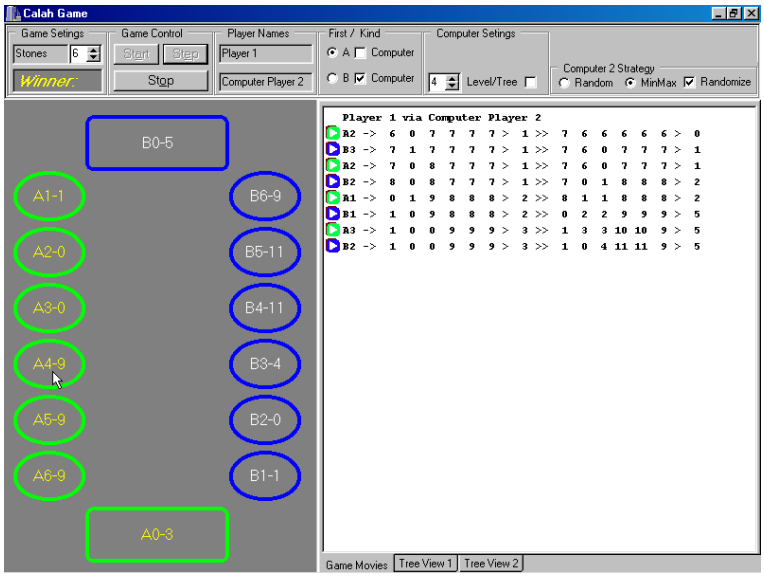
3.1. Програмната структура

На фигурата е представена структура на програмата и основните взаимовръзки.

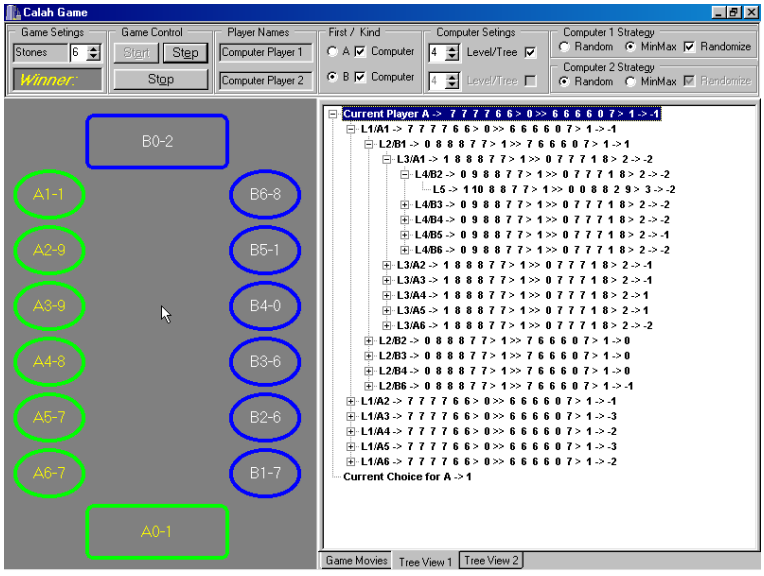


3.2. Програмен интерфейс

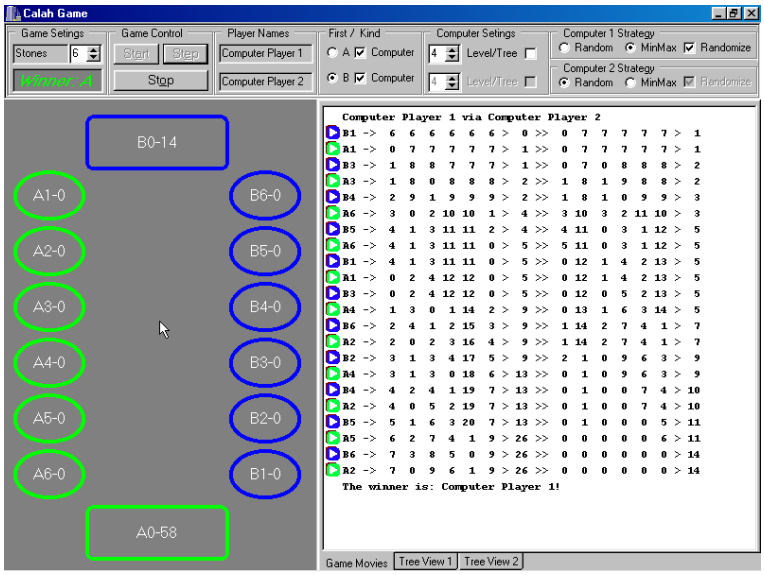
На фигурите е представен интерфейса на програмата в няколко различни режима.



Фиг. 3-1. ***Програмата в режим на игра Player – Computer Player***



Фиг. 3-2. ***Програмата в режим на игра Computer Player – Computer Player с визуализация на дървото на анализа от Computer Player 1***



Фиг. 3-3. ***Програмата в режим на завършила игра Computer Player – Computer Player***

4. Приложения

4.1. "Lunka.h"

```
#ifndef LunkaH
#define LunkaH
//---------------------------------------------------------------------
#include <SysUtils.hpp>
#include <Controls.hpp>
#include <Classes.hpp>
#include <Forms.hpp>
#include <StdCtrls.hpp>
//---------------------------------------------------------------------
class PACKAGE TLunka : public TLabel
{
private:
     int FValue;          // Number of stones
     bool FLKind;         // Lunka or Calah
     TLunka * FNext;      // Pointer to Next
     TLunka * FPrevious; // Pointer to Previous
     int FPOwner;         // Player A -> 0 or B -> 1
     int FPNumber;        // Order number 0 -> calaf, 1-6 -> lunka
     int FSNumber;        // Initial number of stones

     // Check, Set FValue and repaint
     void __fastcall SetValue(int Value);
```

```
protected:

public:
     // Construct and set initial parameters
     __fastcall  TLunka(TComponent* Owner);
     // Initialize move started from current lunka
     void __fastcall GoFirst(int CurValue);
     // Continue move to next lunka
     void __fastcall GoNext(int CurValue,int CurPlayer, bool
                              flag);
     // Continue move to previous lunka - taking stones
     void __fastcall GoPrevious(int CurValue, bool flag);
     // Painting object
     void __fastcall PaintLunka(void);

__published:
     __property Align ;
     __property Alignment ;
     __property Color ;
     __property Enabled ;
     __property Font ;
     // State properties
     __property int Value = {read=FValue, write=SetValue,
                              default=0};
     __property bool LKind = {read=FLKind, write=FLKind,
                              default=true};
     __property int POwner = {read=FPOwner, write=FPOwner,
                              default=0};
     __property int PNumber = {read=FPNumber, write=FPNumber,
                              default=0};
     __property int SNumber = {read=FSNumber, write=FSNumber,
                              default=0};
     // Link properties
     __property TLunka * Next = {read=FNext, write=FNext,
                              default=0};
     __property TLunka * Previous = {read=FPrevious,
                              write=FPrevious, default=0};
     // Handler for Click Event
     __property OnClick;
};
#endif
```

4.2. *"GameField.h"*

```
#ifndef GameFieldH
#define GameFieldH
//---------------------------------------------------------------------
#include <SysUtils.hpp>
#include <Controls.hpp>
#include <Classes.hpp>
#include <Forms.hpp>
#include <Graphics.hpp>
#include <Messages.hpp>
#include <ExtCtrls.hpp>
#include <comctrls.hpp>
#include <Outline.hpp>

#include "Lunka.h"

//---------------------------------------------------------------------
class PACKAGE TGameField : public TWinControl
{
private:
// Timers, staring computer's movies
     TTimer * Timer1;
     TTimer * Timer2;
     // Shapes of lunki objects
     TShape * LunA[6];
     TShape * LunB[6];
     TShape * CalA;
     TShape * CalB;
     // lunki objects
     TLunka * LunkyA[7];
     TLunka * LunkyB[7];
     // Initial number of stones per lunka
     int FPNumber;
     // Current parameters
     int FCurPlayer;
     int FPlayMode;
     // Player -> true, Computer -> false
     bool FPl1Kind;
     bool FPl2Kind;
     // Individual computer player's parameters
     int FLevelPlay1;    // Depth for Min-Max algorithm
     int FLevelPlay2;
     int FStrategyPlay1; // Random -> 0, Min-Max -> 1 algorithm
     int FStrategyPlay2;
     bool FRandPlay1;    // Random choice in equal move
     bool FRandPlay2;
     // Computer algorithm with (false) or without (true)
     bool FPl1Auto;
     bool FPl2Auto;      // tree camputing and viewing
```

```
    // Pointer to List View for listing of movies
    TListView * FListView1;
    // Pointer to Tree View for listing of C1 tree
    TTreeView * FTreeView1;
    // Pointer to Tree View for listing of C2 tree
    TTreeView * FTreeView2;
    // Pointer to Step Button
    TButton * FSButton;
    // Pointers to Edit controls for Player's names and winner
    //text
    TEdit * FEdit1;
    TEdit * FEdit2;
    TEdit * FEdit4;

    // Choice algorithm
    int __fastcall MoveChoice(void);
    // Recursive function for choice algorithm
    signed char __fastcall TGameField::RFunc(int level, int
                          maxlevel, signed char stat[2][7],
    TStringList * List, bool SavePlay, int CurPlay, int
                          FirstPlay);
    // Handlers for Timers (Computer) and Lunki (Player) Events
    void __fastcall Timer1Timer(TObject * Sender);
    void __fastcall Timer2Timer(TObject * Sender);
    void __fastcall LunkyClick(TObject * Sender);
    // Finishing next play event (new computer cycle)
    void __fastcall Timer3Timer(TObject * Sender);
    // Sizing and Resizing functions
    void __fastcall SizeLun(TShape * Lunka, TRect LRect);
    void __fastcall SizeLunky(TLunka * Lunka, TRect LRect);
    // Managing Windows WMSize Message
    MESSAGE void __fastcall WMSize(TWMSize &Message);

protected:

public:
    __fastcall TGameField(TComponent* Owner);
    __fastcall ~TGameField();
    // Spetial public function for accessing from Main Window
    void __fastcall ReNumLunky(void);
    void __fastcall RePaintLunky(void);
    void __fastcall ChangePlayer(int Player);

__published:
    __property Color ;
    __property Align ;
    // Public properties - general
    __property int PNumber = {read=FPNumber, write=FPNumber,
                          default=0};
    __property int CurPlayer = {read=FCurPlayer,
                          write=FCurPlayer, default=0};
    __property int PlayMode = {read=FPlayMode, write=FPlayMode,
                          default=0};
```

```
    // Public properties – Player's kind
    __property bool Pl1Kind = {read=FPl1Kind, write=FPl1Kind,
                           default=true};
    __property bool Pl2Kind = {read=FPl2Kind, write=FPl2Kind,
                           default=true};
    // Public properties - individual for players
    __property int LevelPlay1 = {read=FLevelPlay1,
                           write=FLevelPlay1, default=0};
    __property int LevelPlay2 = {read=FLevelPlay2,
                           write=FLevelPlay2, default=0};
    __property int StrategyPlay1 = {read=FStrategyPlay1,
                           write=FStrategyPlay1, default=0};
    __property int StrategyPlay2 = {read=FStrategyPlay2,
                           write=FStrategyPlay2, default=0};
    __property bool RandPlay1 = {read=FRandPlay1,
                           write=FRandPlay1, default=true};
    __property bool RandPlay2 = {read=FRandPlay2,
                           write=FRandPlay2, default=true};
    __property bool Pl1Auto = {read=FPl1Auto, write=FPl1Auto,
                           default=true};
    __property bool Pl2Auto = {read=FPl2Auto, write=FPl2Auto,
                           default=true};
    // Public properties - visial controls in Main Window
    __property TListView * ListView1 = {read=FListView1,
                           write=FListView1, default=0};
    __property TTreeView * TreeView1 = {read=FTreeView1,
                           write=FTreeView1, default=0};
    __property TTreeView * TreeView2 = {read=FTreeView2,
                           write=FTreeView2, default=0};
    __property TButton * SButton = {read=FSButton,
                           write=FSButton, default=0};
    __property TEdit * Edit1 = {read=FEdit1, write=FEdit1,
                           default=0};
    __property TEdit * Edit2 = {read=FEdit2, write=FEdit2,
                           default=0};
    __property TEdit * Edit4 = {read=FEdit4, write=FEdit4,
                           default=0};
    // Public properties - pointers internal timers for
    // accessing from Main Window
    __property TTimer * ATimer1 = {read=Timer1, write=Timer1,
                           default=0};
    __property TTimer * ATimer2 = {read=Timer2, write=Timer2,
                           default=0};
    //-----------------------------------------------------------
---
// Message Map of managed Windows massages
BEGIN_MESSAGE_MAP
    VCL_MESSAGE_HANDLER(WM_SIZE, TWMSize, WMSize);
END_MESSAGE_MAP(TWinControl);
};
//---------------------------------------------------------------
#endif
```

4.3. *"Main.h"*

```
#ifndef MainH
#define MainH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include <Grids.hpp>
#include <Outline.hpp>
#include "cspin.h"
#include <Tabs.hpp>
//---------------------------------------------------------------------
class TMainForm : public TForm
{
__published:      // IDE-managed Components
     // Top pannel container for parameters controls
     TPanel *Panel1;
     TRadioGroup *RadioGroup3;       // First Player
     TButton *Button2;               // Stop
     ButtoTPageControl *PageControl1;
     // Container for List & Tree Views
     TTabSheet *TabSheetM;           // Sheets of PageControl
     TTabSheet *TabSheet1;
     TTabSheet *TabSheet2;
     TGroupBox *GroupBox1;           // Container
     TGroupBox *GroupBox2;           // Container
     TGroupBox *GroupBox3;           // Container
     TEdit *Edit3;
     TCSpinEdit *CSpinEdit1;
     TCSpinEdit *CSpinEdit2;
     TCSpinEdit *CSpinEdit3;
     TCheckBox *CheckBox1;
     TCheckBox *CheckBox2;
     TEdit *Edit1;
     TEdit *Edit2;
     TCheckBox *CheckBoxC1;
     TCheckBox *CheckBoxC2;
     TGroupBox *GroupBox4;
     TButton *Button1;          // Start
     ButtonTButton *Button3;    // Step Button
     // Splitter of client part of Main Win
     TSplitter *Splitter1;
     TEdit *Edit4;
     TTreeView *TreeView1;
     TTreeView *TreeView2;
     TListView *ListView1;
     TImageList *ImageList1;    // Two Images for List View
```

```
    TRadioGroup *RadioGroup1;
    TRadioGroup *RadioGroup2;
    TCheckBox *CheckBoxR1;
    TCheckBox *CheckBoxR2;
    void __fastcall FormClose(TObject *Sender, TCloseAction
                              &Action);
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall CSpinEdit1Change(TObject *Sender);
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);

    void __fastcall CheckBoxC1Click(TObject *Sender);
    void __fastcall CheckBoxC2Click(TObject *Sender);
    void __fastcall CheckBox1Click(TObject *Sender);
    void __fastcall CheckBox2Click(TObject *Sender);

    void __fastcall Button3Click(TObject *Sender);
    void __fastcall CSpinEdit2Change(TObject *Sender);
    void __fastcall CSpinEdit3Change(TObject *Sender);

    void __fastcall TreeView1MouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y);
    void __fastcall RadioGroup1Click(TObject *Sender);
    void __fastcall RadioGroup2Click(TObject *Sender);
    void __fastcall CheckBoxR1Click(TObject *Sender);
    void __fastcall CheckBoxR2Click(TObject *Sender);
private:    // User declarations

public:           // User declarations
    __fastcall TMainForm(TComponent* Owner);

};
//---------------------------------------------------------------
extern PACKAGE TMainForm *MainForm;
//---------------------------------------------------------------
#endif
```