Programmer’s Documentation

**A picture containing object

Description automatically generatedMineZweeper**

Markéta Sauerová

Univerzita Karlova MFF

zimní semestr 2018/2019

Programování I.

Contents

[1. Specification Breakdown 4](#_Toc13425075)

[1.1. Brief description 4](#_Toc13425076)

[1.2. Functional requirements 4](#_Toc13425079)

[1.2.1. Menu 4](#_Toc13425080)

[1.2.2. Game 5](#_Toc13425093)

[2. Project Structure 6](#_Toc13425094)

[2.1. Intro1 6](#_Toc13425096)

[2.1.1. Global variables 6](#_Toc13425097)

[2.1.2. Procedure Main()[1,4,5,10] 7](#_Toc13425098)

[2.1.2.1. Initializing graphics 7](#_Toc13425099)

[a. procedure Initialise() 7](#_Toc13425100)

[b. procedure LoadFromFile (fileName: string; title: boolean) 8](#_Toc13425101)

[c. procedure MenuButtons() 8](#_Toc13425102)

[d. procedure AnimateTitle (var anim: AnimatType)[6,7] 8](#_Toc13425103)

[2.1.2.2. Maintaining graphics 8](#_Toc13425104)

[a. procedure PressButton(j: integer; menu:boolean)[6] 8](#_Toc13425105)

[b. procedure AnimateTitle2(anim: AnimatType; var i: integer)[7] 8](#_Toc13425106)

[c. procedure UnpressButton(j: integer)[6] 9](#_Toc13425107)

[I. procedure ClearButton(j: integer)[6] 9](#_Toc13425108)

[2.1.2.3. Prime loop 9](#_Toc13425109)

[a. function ProcessMouseEvents (var buttonPressed: boolean): integer [3] 9](#_Toc13425110)

[b. procedure StartGame()[8] 9](#_Toc13425111)

[c. procedure Difficulty()[1,3,4,9] 9](#_Toc13425112)

[d. procedure Instructions()[6] 10](#_Toc13425113)

[e. procedure HighScore()[6] 10](#_Toc13425114)

[I. procedure BackToMainMenuGraphics()[6] 10](#_Toc13425115)

[II. procedure RewriteMenuResolution(j : integer)[9] 10](#_Toc13425116)

[III. procedure DrawBack() 10](#_Toc13425117)

[IV. procedure Back()[1,5] 10](#_Toc13425118)

[2.1.2.4. Finalizing graphics 10](#_Toc13425119)

[a. procedure Finalise()[10] 10](#_Toc13425120)

# Specification Breakdown

## Brief description

MineZweeper is a modified implementation of the classical game MineSweeper, which is a single-player puzzle video game, written in Pascal, using Wingraph, winmouse and wincrt units for better graphical appearance as well as handling user input.

*Original specification:* <Specification-MineZweeper.pdf> .



## Functional requirements

### Menu

1. CloseGraph request

*At any stage, program must react to close graphical window request and properly end itself. It’s meant to be a prime feature for users.*

1. Ban Console

*Forbid the console window from appearance, so that user can focus only on graphical window.*

1. Process mouse events

*Accurately process mouse clicking to prevent the menu from being uncontrollable.*

1. Input reaction time

*Feedback on mouse input from menu buttons should be fast with a small delay, when creating the notion of pressing the button.*

1. CPU usage management

*Don’t use all the CPU time in main loops, checking for the input. Instead use delay function for resting.*

1. Redrawing graphics

*Minimize graphics bugs, while redrawing or overdrawing window.*

1. Reduce flickering

*Reduce flickering of the animated title to the minimum possible.*

1. Execute new process

*Must be able to transfer prime event handler, close graphical window and exit the old process when executing the game, that is written as a separate program.*

1. File handling

*File managing for passing data between individual programs.*

1. Control memory

*Freeing allocated stuff in memory for instance animations, images. Preventing memory leaks.*



### Game

1. CloseGraph request

*At any stage, program must react to close graphical window request and properly end itself. It’s meant to be a prime feature for users.*

1. Ban Console

*Forbid the console window from appearance, so that user can focus only on graphical window.*

1. Loading time

*Reduce time spends with loading and initializing to shortest possible.*

1. Difficulty settings

*Every variable element of the game, that depends on difficulty, must be changed due to settings from Menu output.*

1. Mine distribution

*Distribution of mines must be random and different for majority of newly started games.*

1. Follow game rules

*Square manipulation as well as losing or winning the game should be implemented according to Minesweeper rules.*

1. Mouse accuracy

*The accuracy of clicking on the specific square must be high, avoiding any unwanted squares to be marked.*

1. Mouse click time

*If the mouse button was pressed the delay when processing should be shortest possible.*

1. Timer

*The counting of the time must start exactly when the user presses any mouse button.*

1. Execute new process

*Must be able to transfer prime event handler, close graphical window and exit the old process when executing the Menu or NicknameWindow, which are written as separate programs.*

1. File handling

*File managing for passing data between individual programs.*

1. Control memory

*Trying to optimize occupied memory space, freeing allocated stuff in memory. Preventing memory leaks.*

1. Redrawing graphics

*Minimize graphics bugs, while redrawing or overdrawing window.*

# Project Structure

Project is structured in two main separate programs written in Pascal using Lazarus IDE, **Game01** as the MineSweeper implementationand **Intro1** as Menu and official starter of the whole game. Individual programs are connected by executing each other and exiting their own process.

Along these, minor program **NameWindow**, was also created separately after consideration of possibilities throughout designing the project. Shortly, it serves for obtaining keyboard input from user. Detailed description is situated below.

As mentioned, programs use **wingraph**, **wincrt** and **winmouse** units, which source files are compiled together with code and located in the project folder. Used code will be explained concisely but not in detail. For further usage or interest, it can be found in wingraph unit documentation.

**Sysutils** unit (various system utilities) is also attached and widely utilized.

Programs try to comply with every functional requirement listed above. In this documentation procedures or functions, strictly fulfilling the requirements, will be **marked with number** of the specified one form the list.

For instance, both has the directive **{$APPTYPE GUI}**, which marks the application as a graphical application, so no console window will be opened when the application is run.**[2]**

This documentation is organized according to two main programs and minor third is described afterwards. Procedures or functions are divided in **high-level** and **low-level** design producing distinct blocks handling various parts of the project. In these blocks, used data structures, algorithms, input/output or necessary arguments will be represented and explained.



## Intro1

Overall it is official starter of the game. One of two main programs. Alone, provides Menu support for users with specified functionalities. Start Game executes the Game01 process and exits current one, Difficulty handles files for passing required settings, Instructions overdraw graphical window and loads specified instructions and game rules, Highscore displays highest score reached in individual difficulties along with the gamer nickname.

### Global variables

* anim: AnimatType

*The returned handle of an animation from GetAnim procedure with a bitmap image taken from screen in the defined rectangle. Later serves for animating the title (rotation). Freeing the memory happens at the end by FreeAnim procedure.*

* i: integer

*Helps with preserving animation of the title, which is splitted in two procedures maintained in prime loop (it has to be changeable and accessible from more places). Also used as indicator when should unpressing menu button be performed.*

* colors: array [0..4] of ^longword = ( @red,@orange,@green,@Blue,@Purple)

*Main menu colors, used for simplifying possible changes in different procedures (LoadFromFile,* *MenuButtons,* *PressButton,* *UnpressButton).*

* word: array [0..3] of string = ( 'Start Game','Difficulty','Instructions','Highscore')

and word1: array [0..2] of string = ( 'Beginner','Intermediate','Expert')

*Continuing usage of these strings (MenuButtons,* *PressButton,* *UnpressButton, Difficulty).*

* + buttonPressed: Boolean

*For handling the notion of pressing the button at first menu stage. Used in prime loop and changed by procedure ProcessMouseEvents.*

* bitmap: pointer

*Points to allocated memory block with loaded static image (back.bmp). Used throughout the program, freed at the end or when executing Game01 process.*

### Procedure Main()[1,4,5,10]

The highest-level procedure, that is the only one performed in the main begin…end block, consists of high-level procedures for **initializing graphics** (*Initialise(), LoadFromFile(), MenuButtons(), AnimateTitle()*), **maintaining graphics** (*PressButton(), AnimateTitle2(), UnpressButton()*) and contains **prime loop** for checking if any mouse input has occurred.

For such action, it uses function *PollMouseEvent (mouseEvent: MouseEventType),* returning boolean. When true, the mouse event is returned in the MouseEvent argument. The *MouseEventType* is a record structure, defined in winmouse unit.

When *PollMouseEvent* returns true, MouseEvent argument is processed in function *ProcessMouseEvents().*

In addition, prime loop contains also procedures for menu functionality, StartGame(), Difficulty(), Instructions(), HighScore(), which call depends on position of pressed menu button returned from the function *ProcessMouseEvents()*.

The loop ends if *CloseGraphRequest* is set to true, which is also wingraph feature returning boolean whether user clicked the close button or not.

Afterwards *Finalise()* procedure is used for **finalizing graphics** and *FreeAnim()* is utilized for freeing the allocated memory by AnimatType, when animating the title. *AnimatType* is a record structure defined in wingraph unit.

#### Initializing graphics

##### procedure Initialise()

Uses the wingraph routines for initialization of the graphical window. Also reserves previously obtained *Size* bytes memory on the heap, and returns a pointer to this memory, then loads an image of back icon from back.bmp file there. Back.bmp file is necessary for good functionality of this procedure.



##### procedure LoadFromFile (fileName: string; title: boolean)

If title is set to false, it loads text from a given file until the end of it occurs and displays it on graphical window with stated parameters. Else randomizes given colors from global colors array and use different one of them for every displayed line of the title MineZweeper.



##### procedure MenuButtons()

Draws menu buttons with appropriate text and color from global arrays. For 3D effect is used *Bar3D* shape from wingraph unit.



##### procedure AnimateTitle (var anim: AnimatType)[6,7]

Gets required number of pixels (loaded title) from screen with the function *GetAnim()* and saves them as *anim*, one of the global variables for further manipulation, when creating a rotation animation of title. Also puts *anim* bitmap firstly on the screen for overdrawing the title using *PutAnim()* and parameter CopyPut, defined by wingraph.

Sets the graphics updating to UpdateGraph(UpdateOff) for reducing flickering of the animation.

#### Maintaining graphics

##### procedure PressButton(j: integer; menu:boolean)[6]

Clears the area of pressed button, defined by *j*, and draws new rectangle over it with appropriate color and text from global arrays. Using wingraph’s functions and *ClearButton()* procedure for clearing the rectangle.

It is used to press menu buttons in *Main()* procedure but also difficulty buttons in *Difficulty()* procedure, it depends on the value of the *menu* boolean.

##### procedure AnimateTitle2(anim: AnimatType; var i: integer)[7]

Puts *anim* bitmap on the screen with rotation motion, created by increasing variable i and constant value Pi18 = Pi/18. Firstly by *PutAnim()* and parameter BkrPut and secondly with TransPut, compeling graphics to update now with UpdateGraph(UpdateNow) routine.

##### procedure UnpressButton(j: integer)[6]

Clears the rectangle of pressed menu button, defined by *j*, and draws new menu button with appropriate text and color from global arrays over the area. Using wingraph’s functions and *ClearButton()* procedure for clearing the rectangle.

At last forces graphics to update with *UpdateGraph(UpdateOn)* routine.

##### procedure ClearButton(j: integer)[6]

Clears the viewport of the fixed sized and positioned button, defined by *j*, with white color, manipulating with wingraph’s features (*SetViewPort(), ClearViewPort(), SetBkColor()*).

#### Prime loop

##### function ProcessMouseEvents (var buttonPressed: boolean): integer [3]

For usage and removal of the first mouse event in the queue it uses *GetMouseEvent(mouseEvent)*. Checks if the event is left mouse button click.

If yes, then it gets mouse cursor’s x,y coordinates with *GetMouseX()* and *GetMouseY()* and examines if the cursor’s location is in the area of any menu button.

Returns the position of clicked menu button in integer and constantly works with global variable *buttonPressed*, so the graphics and individual procedures for menu buttons that are invoked in Main procedure.

##### procedure StartGame()[8]

Frees the allocated memory of back icon, closes graphical window and executes process Game01, without inheriting the event handle and exits the Intro1 process.

##### procedure Difficulty()[1,3,4,9]

Redraws the graphical window and creates black colored buttons with difficulty options from global array. After these wingraph functions, *UpdateGraph(UpdateOn)* routine is set for updating the graphics.

Afterward loop for checking the mouse input is executed until one of the options was clicked by left mouse button or the close graphical window request was registered and it automatically returns back to menu (redraws graphics) using *BackToMainMenuGraphics()* procedure or ends.

For mouse input, it uses function *PollMouseEvent (mouseEvent: MouseEventType),* returning boolean. When true, MouseEvent argument is processed in function *ProcessMouseEvents().*

Button pressing graphics is maintained with the procedure *PressButton()*, described above and right argument chosen are passed through text file by *RewriteMenuResolution().*

##### procedure Instructions()[6]

Clears the graphical window with white color- *ClearDevise(),* loads predefined instructions from file *Instructions.txt* – *LoadFromFile()*. Instructions text file is not necessary for the game, but contains useful information for user, which could not be displayed without the file.

Furthermore uses *DrawBack()*, which draws back icon, updates graphics and *Back()* procedure with loop, checking mouse input for going back to the menu.

##### procedure HighScore()[6]

Clears the graphical window with white color- *ClearDevise(),* loads predefined instructions from file *Highscore.txt* – *LoadFromFile()*. Highscore text file is essential for correct functionality of the Game01 program.

Furthermore uses *DrawBack()*, which draws back icon, updates graphics and *Back()* procedure with loop, checking mouse input for going back to the menu.

##### procedure BackToMainMenuGraphics()[6]

Clears the graphical window with white color- *ClearDevise()* and draws menu buttons – *MenuButtons()*.

##### procedure RewriteMenuResolution(j : integer)[9]

Chosen difficulty, defined by *j*, is written in *MenuResolution.txt*, necessary text file for running the program. File is afterwards read in Game01.

##### procedure DrawBack()

Draws a loaded image *back.bmp* in left upper corner of the window.

##### procedure Back()[1,5]

Contains loop for checking mouse input with *the function PollMouseEvent (mouseEvent: MouseEventType),* returning boolean. When left mouse button was clicked it gets mouse cursor’s x,y coordinates and examine if was clicked upon the area of back icon. If so then it returns to menu by *BackToMainMenuGraphics(). Uses delay function for CPU resting.*

#### Finalizing graphics

##### procedure Finalise()[10]

Used for preservation of the graphical window. After close graphical window request occurs, it releases allocated memory with bitmaps and closes graphical window by *CloseGraph()* procedure*.*

## Game01

Second main program Game01 is the modified MineSweeper implementation. It consists of two highest-level procedures, *Load(),* used for all initialization and loading requisite data to memory, and *Main()*, containing the prime loop, which checks for user input and processes it, starts timer and ends if close graphical request occurs.

### Global variables

* bitmaps: array [0..11] of pointer

*Points to allocated memory blocks with loaded static images, used throughout the program, freed at the end or when executing Intro1 process.*

* ended, startTime, first, ex: boolean

*All of these booleans are helping with determining whether some functions should or shouldn’t be executed. Ended controls the prime loop, startTime carries if time should be started, first determines if it is the first use and ex says when to exit, because other process has been executed.*

* rows, cols, count, c: smallint

*Rows and cols are actual values of rows and cols of the grid in played difficulty. C is a constant calculated from number of cols and rows, used for positioning images and text and count is number of mines appropriate also for the grid size. All of these smallints are widely used throughout the program.*

* timeCount: integer

*Counter of the time (score), it’s used at several parts of the code, so it’s convenient and easier to maintain it as global variable.*

* mines, seconds: string

*Auxiliary strings. When drawing text to graphical window happens, string is required, so count and timeCount are used for calculations, mines and seconds for displaying continuously changing text on screen (number of mines and number of seconds passed).*

* grid: array[0..24, 0..16] of STATE

*Maintains player’s visible grid. STATE is an enumerated type used for easier understanding of current square state. STATE = (opened,closed,flaged)*

* grid2: array [0..24, 0..16] of Boolean

*Game grid for mines. True if the mine is located on the square.*

* grid3: array [0..24, 0..16] of integer

*Game grid for calculating and reading numbers around mines.*

Grid arrays don’t have dynamic length according to different grid sizes for different difficulties. Instead they all have size of the biggest one. It’s a more reasonable option.

### procedure Load()

One of two highest-level functions. Serves as an initialization for every structure or construct in the game. It consists of high-level procedures divided in theoretical blocks: **Initializing graphics** (*Initialise()*, *GameStatus(), Menu(), MineCounter(), Timer(), CreateGrid()*) and **initial calculations** (*DistributeMines(), NumbersAroundTiles()*).

#### Initializing graphics

##### procedure Initialise()

##### procedure GameStatus(bitmap: pointer)

##### procedure Menu()

##### procedure MineCounter()

##### procedure Timer(seconds: string)

##### procedure CreateGrid()

#### Initial calculations

##### procedure DistributeMines(count1:smallint)

##### Procedure NumbersAroundTiles()

### procedure Main()