The Game in C++

Final Report

Olga Druzhinina Romanovna

Supervisor: Dr. Tony Y. T. Chan

School of Computing

Faculty of Business and Science
University of Akureyri

Submitted April 2009, in partial fulfilment of
the conditions of the award of the degree BSc.

I hereby declare that this final report is all my own work,
except as indicated in the text:

Signature _____

Date 16/04/2009

Table of Contents

Table of figures

Abstract

In this report I discuss the design of my own game. It is an ambitious project, and it was never expected to be without hiccups along the way. Here I talk about why I decided to create my own game, and why I choose the language for that purpose to be C++ (using Simple DirectMedia Library). I give a detailed description of the game, and get into technical details of the possible solutions I discuss the implementation, evaluation and the future for the game. The game seems to be fully functioning, although one person reported problems, I could not find the same problems and never got a detailed reply from the person. The problem is of an unknown origin since nothing in the code gives a hint at it.

This report was written throughout the allocated project time, and it is very much like a diary that started with mere thoughts and suggestions, and developed into a detailed design, that are not only based on my own decision and ideas, but with the help of other potential players through a short interview. The implementation is discussed later on and the responses of the users of the implemented game are presented.

The report mentions the idea behind my game and how it is related to Pacman – a famous computer game that was developed by Namco about thirty years ago, and that is still played by millions of people throughout the world. Even though I am not trying to create a game that could become popular, I am still learning about the design of the game outlines throughout my project.

The main idea of the project is to simply design and implement a game in C++, because most of the games are written using this language. I start off with no background knowledge in this, therefore the section Background was only filled with knowledge that

I now possess in order to provide the reader with some of the theory behind games before one delves into Design and Implementation sections.

The detailed timeline is presented at the end of the report in Appendix A, and it shows how I have been planning this project. The code is presented in Appendix B. The Appendix C talks about the contents of the CD that is to accompany this report, and points to the Readme.txt file that the user is to read before using the CD. The Readme.txt file is provided on the CD.

Motivation for work

The computer game industry is one of the major entertainment industries today. Although it is very young, it employs thousands of people. Some of the movies nowadays, such as “Resident Evil”, “Lara Croft: Tomb Raider” and “Dungeons and Dragons”, are based on video games. There are even books that are based on computer games, notably “Gabriel Knight: Sins of the Fathers” by Jane Jensen.

Games are not a waste of time! Computer games can educate a person; for instance, sometimes they can provide a player with the knowledge of history or geography unintentionally. A game, whether it is an arcade or adventure, can teach a person to develop such character qualities as patience and tolerance. Games can vastly expand one’s vocabulary, show one how to maintain a family, or let its player be the God and build an entire civilization. And the best part is that games bring fun and joy to all!

The computer games are beginning to show their powers and it’s just the beginning. But with all the possibilities that computer games could have today, they are still being directed at entertainment most of the time. And there are thousands of game

designers, writers, programmers that are capable of much more than what they are creating. The gorgeous graphics does not save the game if the idea is that of another “Doom”. There is more to games than just pure entertainment – it is too obvious to all the game designers, and yet accomplished by few. The game must not have multiple levels of similar obstacles and actions, thus taking up plenty of additional space on your hard drive after installation, yet very often games are developed that way.

Description of the work

The objective of this project is to design and program a game in C++ using Simple DirectMedia Library for graphics. A game will at least use a certain algorithm for it to be reusable, so that multiple levels are not needed. For instance, in a game called “Clue: Murder at Boddy Mansion” the idea is that of starting a new game each time. This particular game places a new puzzle to the player every new game so that the game is new to him/her every time. In my game a method of a new generation of a puzzle each new game will be used as well, so that it will not be a one time game.

The project will mainly concentrate on learning how to program using C++ and learning to use SDL to draw. Here I will design my own game, keeping in mind that the game has to be winnable and losable. Thus I will need to learn how to design a game, writing a short game outline for my game below.

The main aim of the project is to learn to program in C++ and to use SDL, and not learning the game design matter. Regardless while designing the game I will have to keep in mind not only the bare technical issues concerning my programming constraints, but

also the issues concerning the idea behind the game: Who will be able to play it? What skills could this game develop, if any? How is it useful or useless? and more…

Why am I designing my own game? Why not just write a C++ program for another “Pacman”?

The answer is that there exists game creation software, where you can make your own game with little or no knowledge of programming. And that is why not only a programmer must be able to program, but must also be creative. Creativity is always the idea behind any kind of progress.

The functional specification is included in the Design chapter since there is a lot of design issues involved, and the game description is very long.

Related Work

The game that inspired my game would be Pacman. How is my game better than Pacman? It is not! I have a different approach and different idea behind my game. My game is not based on speed, meaning the player doesn’t have to be fast and make fast decisions to win. In my game the player has all the time in the world to make the next move. The only problem is that a player has to be patient and try hard to remember which part of the labyrinth he has already discovered, and to be careful about not making the same move twice – going backwards and forewords. While the human player is trying to get to the shining diamond, the computer player can already reach the diamond by then. So if the human player is paying attention to his opponent’s moves he can if convenient follow the computer player’s path.

Unlike the game of Pacman, my game will train eyesight, keeping the eyes concentrated on the dark field that is the unseen labyrinth. Which each new game, the player's orientation in the dark labyrinth will be better. While Pacman trains the player to make fast decisions, my game gives a player the time to consider his moves thus helping the player to become patient and considerate. Another good feature that my game has is that even if the computer player grabs the diamond the games is not over yet, the human player still has a chance to win. As you know, in Pacman the human player loses the moment a bad guy runs over him.

Background

File Structure

Once the game has been designed in terms of game functions and classes, the file structure of the game is decided upon. There are generally two types of files for each class – header type of file (or include file) and the source type of file (or cpp file).

"The .h file holds the class definition and the inline member functions. The .cpp file contains the member functions that are too big or too complex to be inline." ([7] *Creating Games in C++: A Step-by-Step Guide* by David Conger).

In this report you will need to know this since I will discuss both files – header file (images.h and engine.h) and source (OP_Maze_RP.cpp).

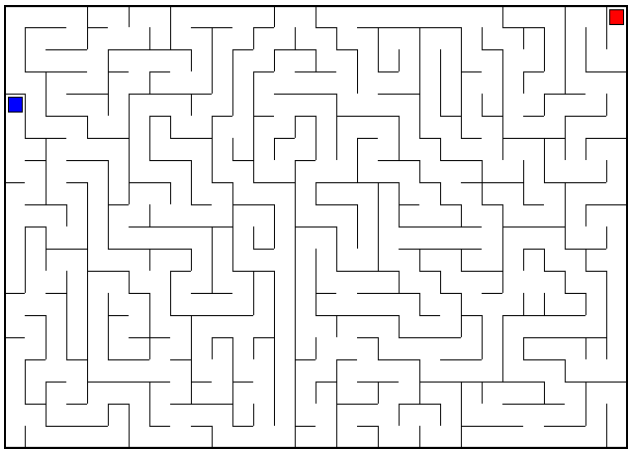
Labyrinth and the Shortest Path

A labyrinth in my game is built based on a one-solution labyrinth. And to get from point A to B would be easy if one didn't have to wonder around going backwards and forewords. So finding the shortest path and following it is what the computer player

will be doing throughout my game. Therefore it is important to have a precise picture of what kind of labyrinth hides in the dark. Below is the picture that inspired the construction of my labyrinth. The filled squares represent cells A and B in the labyrinth. The shortest distance between A and B has a unit measure of cells.

Figure 1 Example of a labyrinth

(source [9] in References)



The idea behind the construction of my labyrinth in my game is similar, only not so complicated.

The first solution and simplest one would be to find a path from A to B, set it to "shortest" variable. Then find another solution and see if the number of cells is less than in the solution before; set it to shortest if it is a shorter path, and then carry on like so till you are left with only one shortest path. Of course, the repeated paths need to be taken into account since the maze itself has a very specific structure – if there's a dead end, you shouldn't check for it every time, otherwise it will turn out to be too time consuming. From there the algorithm could be developed into a more advanced one. A breadth first search algorithm is often used to find the shortest path in a labyrinth.

Game engine

What you need to know about the game engine is that it is a core of the game – all the classes and functions for graphics, sound, artificial intelligence (such as shortest path finding discussed above) managing, and many more. The main source file cpp only uses the constructors and functions of the game engine.

Simple DirectMedia Library (SDL)

The SDL Library offers a lot of functions and operation, but I will only make use of:

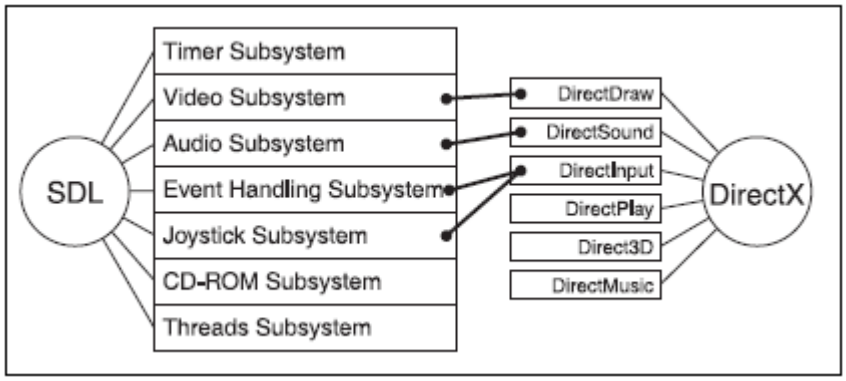
SDL Video: “Like DirectDraw, SDL’s video subsystem deals primarily with surfaces but also has structures for rectangles, colors, palettes, and overlays” ([8] *Focus on SDL* by Ernest Pazera, page 22).

SDL Event Handling and Window manager: helps you deal with events, such as mouse clicking or keyboard buttons pressing, in a way you want to handle or respond to one or another action regarding, for instance, user’s input.

SDL Timers: “precise timing is important to any high-performance application, such as a game.” ([8] *Focus on SDL* by Ernest Pazera, page 24).

Figure 2 SDL library

([8], *Focus on SDL* by Ernest Pazera, page 21)



Design

The game involves two players: the human player - the Good Guy (you), and the computer player – the Evil Dude.

Both of the players are travelling along the labyrinth. The labyrinth cannot be seen by a Good Guy. It's a lot like in real life – if you get inside the real labyrinth you will not be able to see the road even if you are using a flashlight.

The aim is to get to the diamond and get out of the labyrinth first.

This will not be easy for several reasons. First of them being the presence of the Evil Dude – the computer "knows" the labyrinth trying to find the fastest way through it first to the diamond, and then to the door out of it.

The Design might change a little as logical improvements arise in my head concerning its usability. But the changes are to be insignificant, so below is the main plan for the design.

Menu

And now the first thing that the user will see after loading the executable file will be a little window with the following Menu on it: "Rules", "Play", "Exit".

The button “Rules” will redraw the window look so as to add additional information in the upper right corner of the window.

The button “Exit” will make the program abort itself.

And the button “Play” will launch the game, or in other words it will redraw the window with the script running the game, and if you try to close the window manually (that is before the game ends), you will be taken back to menu.

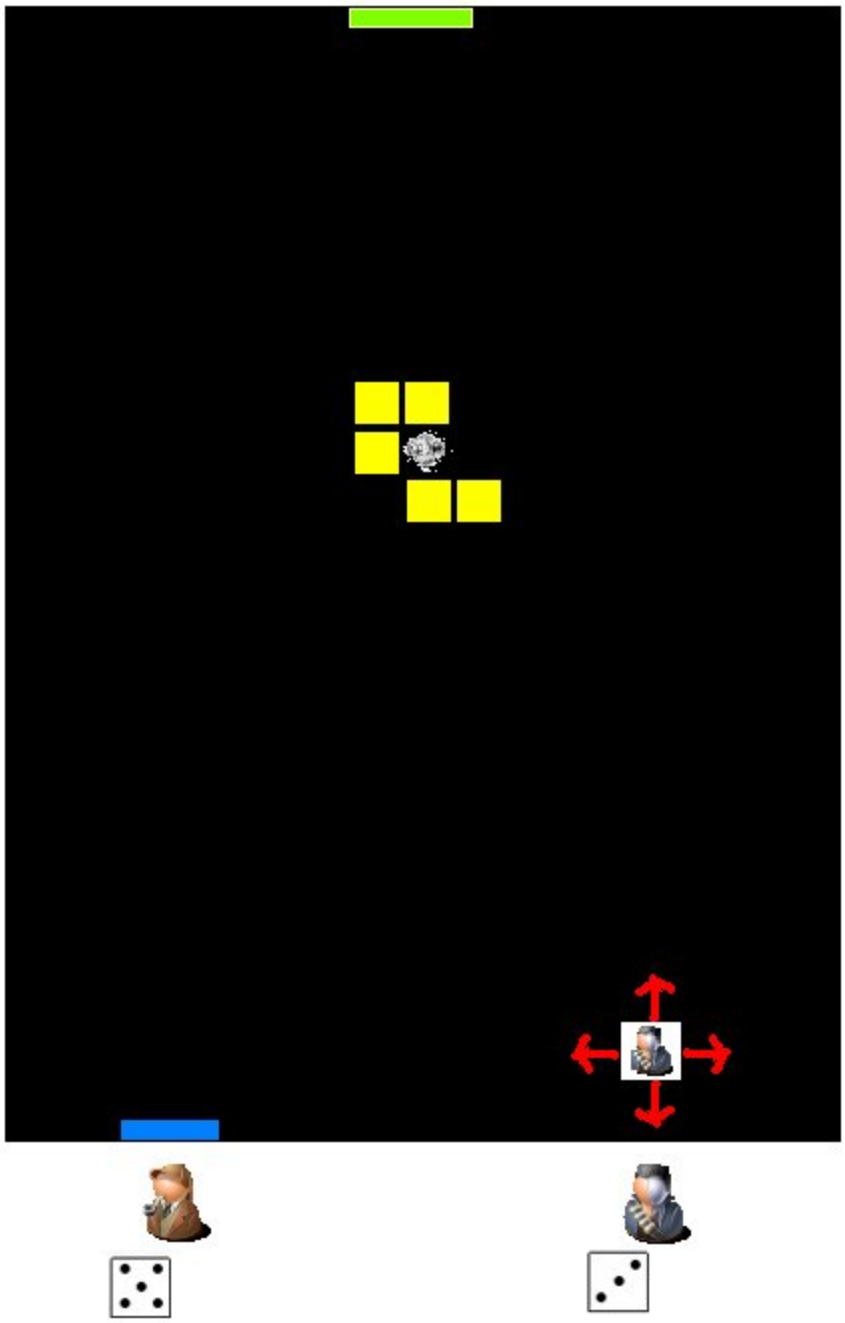
Moving

The Good Guy (you) always goes first. To make a move, you will need to click on the die that is to be situated next to your character. The number you will get on the die indicates how many steps in the labyrinth you will make. If you hit the wall, you do not lose your step. Once you take your last step, it is the Evil Dude (computer)’s turn.

The Labyrinth

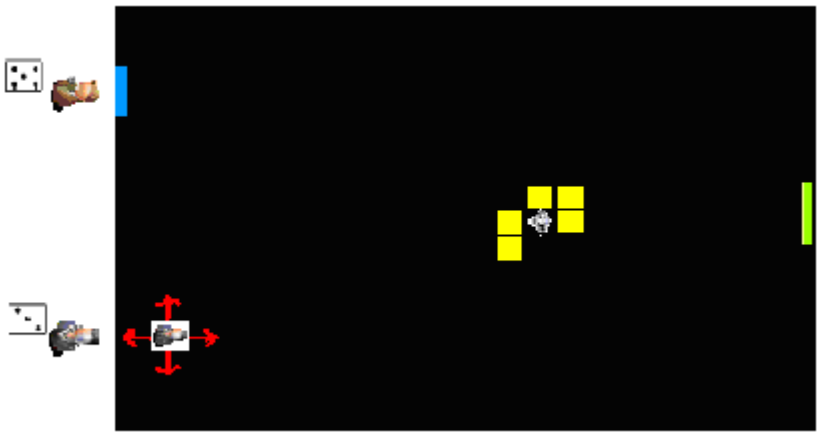
The labyrinth is generated each new game, making the game reusable, so that it doesn’t have to have predefined levels. The invisible cells make up the labyrinth. The cells are only visible around the diamond – diamond behaves like some kind of a light source (like a flashlight), that will show you if the path around the diamond for exactly one cell around you. Below you can see the diamond is still untouched.

Figure 3 Design interface



The picture above was the first draft, this has later proved itself to be of uncomfortable design, so the picture is rotated like so:

Figure 4 Interface alignment



And this design will be the main idea of how it will look at the end, perhaps some changes may be done as to change the characters' looks – here it's just some icons. To give the game a certain mood, I might change the look of the icons at the end, but that of course does not matter as of now.

The labyrinth is a little tricky: the idea behind it is that from the diamond to the "Exit door" (to the right) there is only one solution, but from the "Enter door" (for each player the first starting position) to the diamond, there is one or two solutions since there are two players. To make it more interesting to play this game for the human player, I decided to randomly generate two possibilities – either there is one solution for two players, meaning that they will meet somewhere along the way, or there is two solutions – one for each player, meaning that each one of them is going straight to the diamond using a different path and the players don't meet along the way.

The Diamond

The diamond is placed in the middle of the screen for the convenience of the game, so that the diamond divides the distances between the starting point (the "Enter Door") and the finishing point (the "Exit Door") into two equal parts. To grab the

diamond the player needs to walk over it that is to either get on the same space/cell or to cross that cell and move on, thus already moving with the diamond in your pocket. The cells around the diamond that are visible are in fact bricks. Bricks here, as in real life, make up the walls, meaning you will have to go around them to make your way to the "Exit door" in order to win the game. And since neither of the players have the ability to fly over the walls like the magical creatures in today's modern games often do, the players will have to use the way through the empty (black) cells.

If you do not grab the diamond first the game is not over yet. All you have to do in order to get the diamond from your opponent is to simply walk over it, and you will again hold the diamond. But if you did get the diamond, hurry to escape with it out the "Exit Door" because the Evil Dude will sure try to steal it from you at any price. Once you get the diamond it must appear in the panel below to signify that you are the current owner of it, like so:

Figure 5 Player: die and diamond

The Evil Dude

The Evil Dude has an attitude. It is to be programmed to:

1) find the shortest path to the diamond;

2) in procession of the diamond, find the shortest path to the "Exit door";

3) in case the diamond is at the hands of the Good Guy, try to either steal the diamond or get in your way and block it in order to prevent you from getting to the “Exit door”. Then, if you are trying to escape the Evil Dude and are not going towards the “Exit door”, but instead are wandering around the labyrinth, the Evil Dude should start moving towards you to get the valuable diamond.

Sounds logical and simple, but the question is: In case it blocks your way to the “Exit door”, should it stay very close to the exit door (one cell away from it, for instance)? Will it make it difficult for the Good Guy to win? Does it pose the risk for the Evil Dude to lose such a game of luck? As a human player you can try this trick but that would be like gambling… The final decision concerning this can only be made either during the programming stage or maybe even after the game is nigh complete and I run the game’s tests. I am not able to imagine the whole game, but this issue will not be left unsolved.

The other thing is that the labyrinth should be interconnected in order to provide the players with twists connecting their paths.

The extra panel

There will be an extra panel on the game screen that will notify the user of how many steps he/she has left, and whose turn it is at the moment.

Abstracts from interview

I have asked all the people I know and some of them I didn’t know up till now, whom I consider to be the potential players, and who were interested in providing me with their questions and suggestions, about the design of the game.

Below are some suggestions that I found to be most interesting and that influenced the design, so I added the following to the design as extra features after careful consideration:

1) I will try to make a complicated labyrinth as long as it doesn't make the game impossible to win.

2) Because the game cannot be won without going through the portal door with diamond, the question is: should the portal door be unavailable while the player does not hold the diamond, and only become seen when the player does grab the diamond. I have decided to let the portal door always be displayed since the door is always situated on only one fixed cell, and it could be annoying to see the door blink every time the player is robbed of the diamond by the Evil Dude. But I implemented the feature of the portal door being unavailable if the player does not hold the diamond into the game, because logically the door is indeed the portal. I wrote the code and left it in comments' tags as additional feature for the future use.

3) I decided to implement the cheat into the game for easier testing of the game for me and my teachers. The user is not supposed to use it unless he or she is completely lost and needs serious help. There won't be any cheat codes of any sort – I will simply make it possible for the player to click on any cell on the screen during his/her turn, and the move will be then automatically made using the same function the Evil Dude is using (that is finding the shortest path to the right cell).

4) There is only one character for the computer player, and one – for the human player. Implementing other characters is unnecessary for now and does not change the idea behind the game. My project is directed at implementing the idea of the

game in C++ with SDL library. In the future work, different characters could be used for the extended version of the game.

5) The game will be a two-dimensional game. There is plenty of software available to help one to design a 3D character, but I am afraid I don't have enough time for that, and it needs more than just programming skills to implement a 3D character into a game, but also graphic designer's skills (which I unfortunately am lacking). But it is possible to use 3D icons instead of using DirectX 3D or OpenGL 3D, only again I cannot even draw in 2D, so maybe some other time in the future if the game turns out to be of any appeal to its players.

6) There will be no music or sound effects, because it is not very necessary for the project and not so difficult to implement. There is two types of interaction with the game: the one involving keyboard and the one involving mouse.

Step-by-Step Design

(the technical details)

Why do I choose C++? Simply because around 70% of all the games today are being written in this language. Why SDL? Firstly my supervisor told me that I should use primitive graphics, and not worry myself about such libraries as DirectX or OpenGL. Secondly after searching the forums for the right library, I often found that SDL is the common simple library to make things that I want happen. I tried it. It worked. I am happy. What compiler did I choose and why? My ex-compiler Dev-C++ turned out to be difficult to configure with the SDL library due to the fact that I couldn't find the right tutorial on how it should be done, so I decided to install the Visual Studio C++ 2008

Express Edition because I found an easy to follow tutorial to set the SDL library with VC++ up, and of course not to mention the fact that VC++ was popular and free.

For those who can read C++ code, the code is presented in Appendix B for the files OP_Maze_RP.cpp, images.h and engine.h. The code is presented in the appendix solely for the deeper understanding of what is happening in the game. It could receive changes later, so it is not to be treated as the final version. The code is also not very well presented as for now, but in idea it is almost complete and is the one used in a fully working game. The most important header files are images.h (the functions of making life easier while working with SDL) and engine.h (engine of the game, the players' classes, graphics, etc.). These will be included in the OP_Maze_RP.cpp file.

The main and most important functions in OP_Maze_RP.cpp are:

Menu():

This is a function that specifies how the menu is set. Well, the idea behind it is pretty easy. We have four pictures of the menu in its four states – one is the original, the other three is for when the buttons get pressed: “About”, “Play”, “Exit”. Remember, I talked about them before in the beginning of the Design Chapter? Well, now the pictures will be used/redrawn as the mouse is pointed over a particular/corresponding area on the screen and is clicked. In case of the “About” area, the writing in the right upper corner of the screen will be generated.

ThrowCub():

The function for the waiting of the user's (human player) next throwing of a die. This function’s main purpose is to catch an event, a mouse click on the die event to be

precise. Here only the MOUSEBUTTONUP of the SDL library event type is being taken into consideration. The MOUSEBUTTONUP only considers the SDL_BUTTON_LEFT (left button on the mouse), and ignores if the right button of the mouse is pressed. After the user clicks the die, the writing in the lower panel that is next to die will be erased. The function returns false in case the game has been aborted.

ProgressUserPlayer (player user, field labirint): takes in two parameters. There is a player and a field constructors involved, which are to be coded in a previously mentioned header file engine.h.

This function is about the human player's turn and the moves he/she makes inside the labyrinth.

Notice that to avoid major confusion of a defined in the program word labirint and the english word labyrinth, so that the comments are clearly referring to an english word rather than to a defined variable. I also define the word "cubic" with reference to the word "die" to avoid the confusion of the latter word's twisted meanings.

The function returns the messages of the sdl types:

-1: the game was closed

1: the player has won

0: the player has finished his/her turn/move(s)

How is it known that the solution of the labyrinth was indeed found?

The answer is this: This will be done by comparing the coordinates of the coordinates of two players to the coordinate of the predifined cell of the exit door.

The variable that holds the state of a winner will then be used to generate the message with the winner's character picture upon it, thus notifying the human player of who won the game. The creation of the message and redrawing of the screen with this message is NOT a part of this function.

This function catches the events of the human player's arrow keys pressings. The human player is using arrows to move the character like a piece on the game board. There are four arrows SDLK_UP, SDLK_DOWN, SDLK_LEFT, SDLK_RIGHT. According to which one is used, the character is drawn in a new position after each step, that is as it moves. Each time after re-drawing the character, the state of the game is checked (win or lose that is). Then the changes are renewed on screen.

Now with the mouse events. The event is generated as the mouse button gets pressed. The mouse controls are again depending on the coordinates. For example, in case the click has happened in the panel under the labyrinth is, the situation of the toolbar is taken into account. The scrolling is also handled in this function so that it doesn't get in the way with the game.

ProgressCompPlayer(player computer):

It's computer's turn now…

This involves calling throwing of a die function that is called cubic(), and its purpose is to generate a random number on a die (ranging from 1 to 6).

Then the function for calculating/deciding upon a shortest path and moving along it towards the exit door is called. This of course has to be done in view of the number of steps that were generated using a die.

The status of the game (whether it’s a win or a loss) is checked by the computer after its move. The computer player awaits the user’s turn.

_tmain:

Takes in two attributes. This function is specific and it deals with several important issues. I will not get into too deep details of it, all the details can be found in Appendix B – the one with the code.

Mainly what this function does is setting up a video regime, all the details such as screen size, name of the window, pictures… Pictures are converted to boost speed of the appearance of the graphics.

There is the declaration of the object of the class "field" (the area of the labyrinth) that is taking place in the function and the labyrinth is built. This function does not include the code of the class “field” construction – it is a different class the constructor of which is in the header file engine.h.

The two players are also created as objects of another class called “player”, the constructor for which is again in the header file engine.h. The players are then drawn to the screen.

And finally the move is to be made. There are two switch loops one inside the other constructed like so:

For the user:

Case -1: The exit has been fixated.

Case 1: The user has won, start new game.

Case 0: The user made a move and is now awaiting the computer player's turn.

For the computer:

Case 1: The computer player has won, start new game.

Case 0: The computer player has made its move and is now awaiting the user's turn.

The above basically describes the behaviour of a game and the relation of the moves/turns between the human and computer players.

In the header file **images.h**:

Firstly the pictures are declared, but they will be converted in the future by the function in the file OP_Maze_RP.cpp in order to make life easier for the graphics, as mentioned earlier.

In images.h I will omit some of the additional complicated functions and operations that the SDL library offers.

The header file images.h has everything to do with drawing as the name of the file suggests. The functions defined in it draw pictures, text to the screen with predefined fonts (thus slowing down the performance of the game).

In the header file **engine.h**:

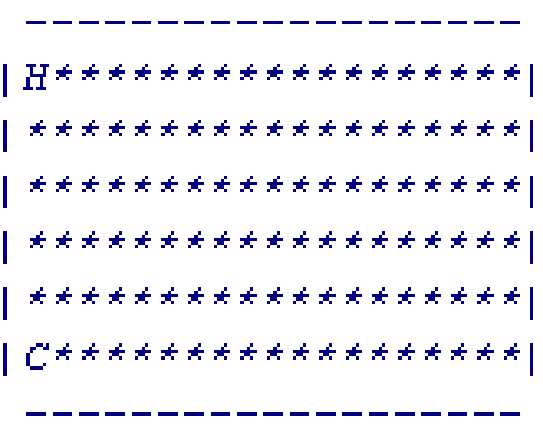
The control over the game is implemented. Classes are created – for labyrinth, for players, for walls and squares. Here are the functions to manage graphics, like drawing of the screen and the path, the die changes and the diamond that act as a source of light, and the exit door that is supposed to behave “like a portal”.

The main solution to such problems is to build a matrix, which is of course built to generate labyrinth. Using this matrix you could manage and understand what is to be

done with the drawing and redrawing of the characters, clicks of the mouse on the “board” of the game, that is the panel where the game is taking place, etc.

Two of the cells on the field are not to be touched, since they are left for the two players. These cells are situated vertically to each other, like so:

Figure 6 Labyrinth structure



Above H stands for human player, and C - for computer player. As you may notice, the game “board” is inverted so that it fits the screen. For the labyrinth to be a little longer, I decided to place the door in the left lower corner, but the diamond will still be placed in the centre.

Class “square” represents a cell, and has the functions of being drawn, and the Boolean function to find out what type of cell it is – whether it is a wall or not. Thus, logically there are then classes of the square walls. There are three colors of walls – still not sure whether I will keep it this way, but for now it is convenient for me to use three types of walls – the one that goes around the labyrinth, and the two that lie within it. Perhaps changes will be made and only one type of wall will be used in the final version.

For the class field the matrix is created for the labyrinth as mentioned before (see Figure 6). Then there is also auxiliary matrix that gets created for reference points/cells. The getrandom function, as the name suggests, gives out a random reference point (for the walls' generator). To create the walls in the labyrinth, we first need to get a random reference point/cell to distinguish between the clear path and the one with wall.

The drawing of the labyrinth is then possible through the matrix's cells. We draw all the walls, the portal door, the players, etc. We too draw the extra panel, that guides the human player – I call it toolbar. It says "Steps:" (the overall number of steps the user has taken throughout the game) and "Steps remaining:" (how many steps remain for the user to finish a certain move) on it. For example, it could say "Steps: 34 Steps remaining: 2", and it would mean that the user has taken 34 steps already, and it is now his/her turn (the die has been thrown already), and the remaining steps for this turn is only 2.

The functions for the calculations of steps remaining, the number on a die, drawing a die, and such are all very straightforward and need no explanation.

The calculation of the shortest path is managed through a simple algorithm that compares different ways using the information of the matrix, and thus chooses the best/ shortest path for the player to a particular/specified cell.

The events are caught:

If ESC is pressed, the exit happens;

If at the end of the game any keyboard button is pressed, the menu is redrawn (the very same at the beginning when the application is launched);

If the arrow buttons on the keyboard are pressed during the user's turn, the character moves in a specified direction (depending on the arrow button pressed). If the character faces the wall, the number of steps is not diminished, and the character can press the button again.

If the mouse button is pressed on the die when it is user's turn, the die is generated and the corresponding picture of a die appears/gets redrawn;

If the mouse button is pressed on one of the cells on the board during the user's turn, the piece/character is moved towards that cell as many steps as he/she has remaining.

Implementation

The Technology Platform:

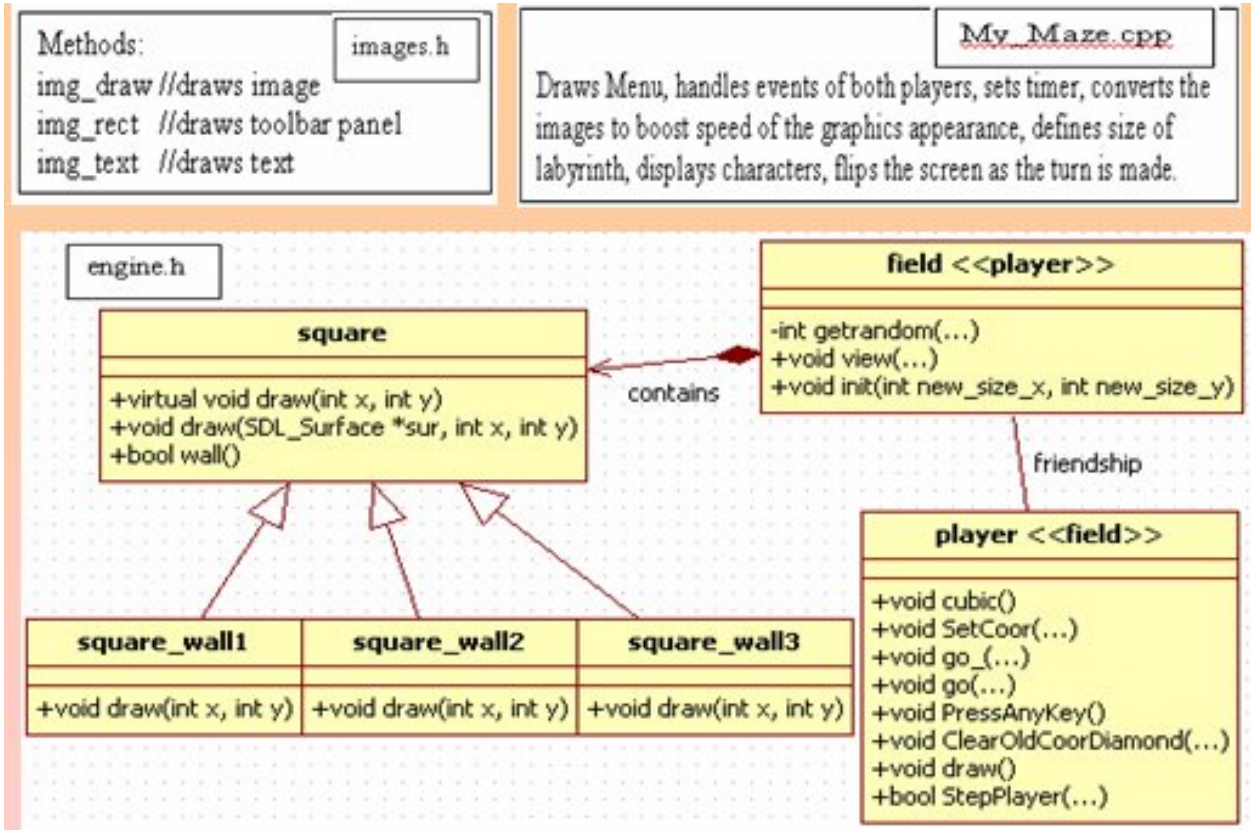
About software: I used Visual C++ 2008 Express Edition compiler. The game may therefore require Microsoft Visual C++ 2008 Redistributable Package (x86), but as far as it has been tested on school computers and an old laptop it did not. I have had some trouble managing with the compiler and configuration, and that is why it is uncertain to say if or if not the game requires the package indeed.

Hardware: My system is Microsoft Windows XP Professional, v. 2002, service pack 2.

The game runs on Windows XP.

The Microsoft Visual C++ 2008 Redistributable Package (x86) installs runtime components of Visual C++ Libraries required to run applications that were developed with Visual C++ on another computer that does not have Visual C++ 2008 installed.

Diagram



I use header files from SDL library by Sam Lantinga (the creator), such as SDL_events.h, SDL_timer.h, etc.

I use methods I implemented in images.h for drawing in engine.h and OP_Maze_RP.cpp. I use engine.h classes and methods in OP_Maze_RP.cpp.

The explanation of the diagram:

The word “virtual” means that the derived classes can override the member function of the class (in this case the square class) in order to have a different functionality. For instance the square walls use different images, therefore the virtual method draw is overridden, while the functionality of other methods that square walls inherit from the class wall remains the same.

A friend function is a special function which can access the private members of a class. So player can access the private members of the field class.

Void view(…) is for drawing of the screen with the defined size of labyrinth – for future use.

Void init(…) is for labyrinth generation.

Void cubic() is for throwing the die.

Void go_(…) calculates the shortest path from point A to B.

Void go() is for mouse control while user clicks to go_(…) [called a cheat].

Void draw() is for drawing of the character.

Bool StepPlayer(…) is when player makes a move (returns true).

Key Implementation Issues

Labyrinth algorithm is generated each new game, making the game reusable.

The other thing about labyrinth is that from the diamond to the portal door there is only one solution, but from the start of the labyrinth to the diamond there is one or two solutions since there are two players. The issue was solved by making the program only generate the conditions above, meaning that the labyrinth can have only four fixed places

– one for the human player, one for the computer player, one for the door, and one for the diamond. Fixed meaning it is defined those labyrinth cells are not walls. The labyrinth does not take the portal door into account, meaning that the path (solution) can actually go through the door.

The labyrinth is generated each new game, so that the game doesn't have to have predefined levels. The invisible cells make up the labyrinth. The cells are only visible around the diamond: diamond behaves like some kind of a light source (like a flashlight), that shows the path around the diamond for exactly one cell around the current holder of it.

The idea behind it is that from the diamond to the portal door there is only one solution, but from the start of the labyrinth to the diamond there is one or two solutions since there are two players. To make it more interesting to play this game for the human player, I decided to randomly generate two possibilities – either there is one solution for two players, meaning that they will meet somewhere along the way, or there is two solutions – one for each player, meaning that each one of them is going straight to the diamond using a different path and the players don't meet along the way.

Computer player's AI was an important aspect of the game.

Computer player must find the shortest path to the diamond; in procession of the diamond, find the shortest path to the portal door; and in case the diamond is at the hands of the human, try to either steal the diamond or get in the opponent's way and block it in order to prevent you from getting to the portal door. In other words, the computer player uses the shortest paths – for the diamond, for the user, and for the door.

Screenshots

Screenshot 1 (game 1)

The die generated number 4, but after the human player took one step, he now only has 3 steps left to make during this turn. The human player made 9 steps overall.

The diamond is still in place, i.e. not taken by anyone.

Screenshot 2 (game 2)

This screenshot is from another game.

Here the opponent is left behind, and the human player has the diamond (right corner below), and 5 steps to make for this turn. Both players will now move towards the portal door in order for one of them to win.

Screenshot 3 (game 3)

This screenshot is from yet another game.

Here the opponent is left behind, and the human player has the diamond, and is expected to make the next move towards the exit. Meanwhile the opponent (computer player) is trying to block the exit portal door in order to catch the human player and steal the diamond.

Screenshot 4 (game 4)

This screenshot is from another game.

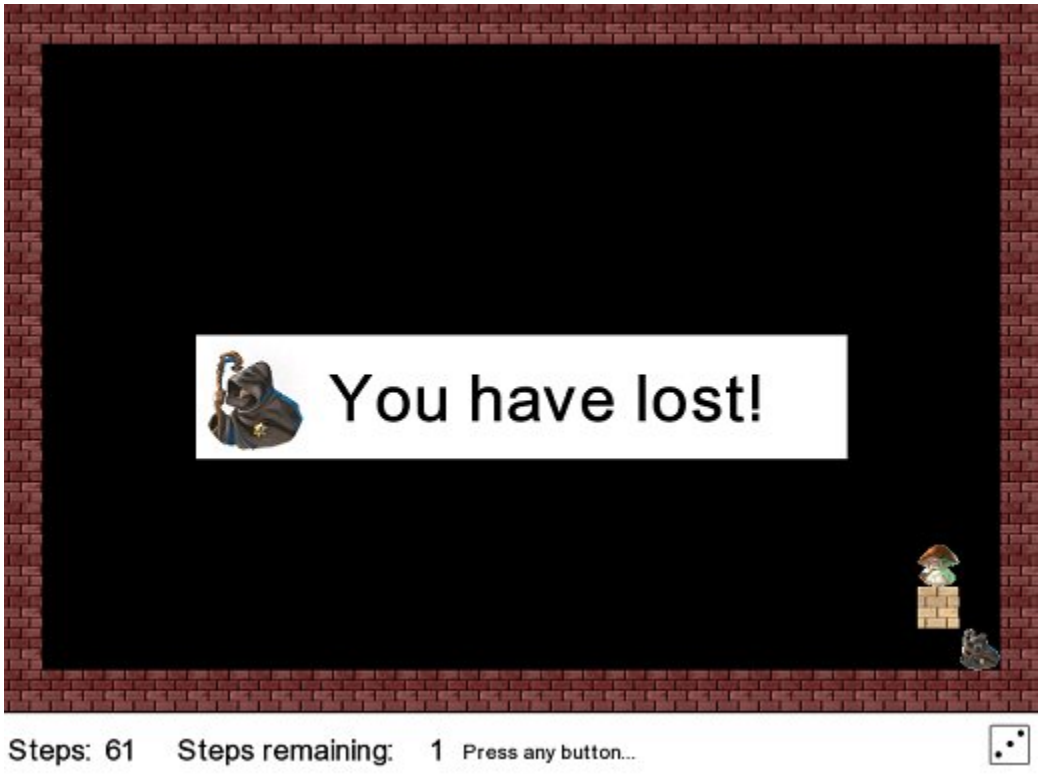
Here the human player is left behind, and the computer player possesses the diamond since it is not displayed in the toolbar panel (the one below the game play). The diamond acts as a light source, so that the human player still has a chance to catch up to the computer player by remembering the path that is around the computer player.

Screenshot 5 (game 5)

This screenshot is from another game.

Here the human player is only a few steps behind, and the path to the portal door is very clear. If the computer player rolled a die of 1, then the possibility of winning for the human player would be very high.

Screenshot 6 (game 5)



This screenshot is from the same game as screenshot 5.

Here the game was won by the computer player. Notice that the direction on the toolbar panel reads “Press any button…” on the keyboard, instead of writing “Awaiting your turn…”.

Evaluation

Tests that I performed while implementing of the game included functionality, compatibility, whiteboxing (static testing, fault injection, code coverage).

While the game was being implemented (in C++) I have tested it a lot during each phase of it. I have been mostly concerned with testing the computer player’s moves and the labyrinth itself. The first labyrinth had only one solution and one (human) player. I

have tested the screen's drawing and re-drawing. I have used the debugger while running of the game.

Functionality testing was done to figure out the general problems within the game itself or its user interface. Report: stability (the game crash does not crash), correctness of game mechanics (maybe the diamond is not a good example of light source, etc.), and integrity of game advantages. The game is difficult to win though.

Compatibility testing was done to find out if the game runs on different configurations of hardware – my computer at home, my mother's computer, school computer, library computer and supervisor's computer. The game ran on all, unfortunately I did not have a chance to run the game on more computers (Operating Systems), therefore I have concluded that the game does run on Windows XP.

In order to blackbox the game, an extra feature had to be implemented. A cheat of human player using the shortest path by clicking (with the left mouse button) on the cell, the player wants to go to. I have done some of the blackboxing by tesing the game while running other applications on the system; testing the game after it has finished again and again; testing the game after moving the folder of the game; etc. The blackboxing was done by playing the game many times using a cheat function mentioned before. I have tried playing it in a normal mode (using keyboard) many times under different situations, but it was very time consuming, thus I have not had enough time to play the game for many times.

Soaking was done to see if the game runs for a period of hours (days) in various modes of operation. Soaking does not involve playing the actual game. The purpose of soaking is to detect the presence of memory leaks or rounding errors that

manifest over time, resulting in system instability or odd game behavior (a definition from wikipedia).

Other user evaluation, where I will ask the volunteers (gamers) via email and forums to play my game and see what flaws they spot, is discussed below:

I have asked some of the users in different forums to play the game and tell me if there is anything wrong they find with it. Most people said it was very difficult to win the game without the use of cheat. One person has reported that there was a problem of redrawing screen sometimes after one quits the game, and without exiting the program, starts another game. The exact problem was with the human player character image being stuck on the screen at the exact place where the previous game was finished. I have blackboxed the game, but this has never happened to me. I asked about the platform and operating system of the user, but never got a reply. Generally it was very difficult to get people to play it since it does not have an interesting plot and is two-dimensional. Out of 15 players, only 10 said the game was running fine. The other 5 complained about the user interface, and did not like to play on a pitch-black screen. Out of the 10 players, only 3 stated they enjoyed the game, the other 7 did not like the fact that it was too difficult to win the game. The 3 players who in fact did enjoy the game stated that their enjoyment lasted only for a limited time, later they all got bored.

As a programming project, I believe it was good. But it needs a lot of work in order to enter the market. Many changed would have to be made to the game, but fortunately the engine of the game wouldn’t need much modifications unless the idea of the game would have to be changed.

Conclusion and Future Work

The game was designed and implemented in C++ using SDL, was tested and is working properly. It does have some minor problems with design issues though (difficult to play).

Future for the game:

- The game could be made in 3D.
- There could be more players (more than two) added to the game.
- There could be more player characters for both computer and human players added in the future. For computer player, there could be more characters with different artificial intelligence abilities; for human player there could be more characters with different attributes, or advantages and disadvantages.
- Different levels of difficulty could be implemented. With the current level of difficulty set to the highest. The easier levels could have more light spots on the screen for the human player easier to make guesses about the labyrinth.
- Extra objects, such as a diamond, with different properties could be implemented.
- The game could be made into a board game. This could be done by placing square magnets together thus making a labyrinth, then putting the dark carton board above the magnets, and placing the figures – the figure cannot move out of the range of magnets since it will always stick to the path.

But the figure could get into a trap in the labyrinth, because not every path is a solution.

It is quite difficult to play because the computer player is too smart and knows the labyrinth. It is winnable of course, but it is a bit difficult – a way to improve it can be to make more computer player characters to choose from, each having unique advantages and disadvantages (rather than having one that is too intelligent).

If I had more knowledge in the game programming in previous semesters, then I would make different design decisions, such as either making a computer player's AI less advanced or giving a human player extra privileges over computer player.

I was down to trying to make the game according to the standards of the difficulty of programming in order to pass the course. If I had been thinking about this project as it was a real-life project, which would need to be sold, I would totally reconsider my decision. Only when the game was finished, and I coincidentally was learning about game programming at the same time, I realized how the game could be improved without ridding the project of the programming abilities.

Personal reflections

Out of all the projects throughout the whole three years of studying Computer Science, I thought this was going to be the most interesting one since I had a chance to do it all by myself. After I designed the game I thought that the most difficult part would be the implementing it in the new language for me.

My expectations were very high as I was writing the code. Every time I would be writing the code for homework I would congratulate myself upon its finish and not think

any further. But after completing the code, at first I was very disappointed since after all the code was written and yet I got negative reviews from users of the game, and I first blamed it on the bad design which was due to the fact that I never studied game programming. But later on I realized that the whole thing could be improved by logically deciding upon the game plot, and coming up with different characters, objects, and attributes in the game.

I am now very glad about the fact that I can change the game by merely dividing the abilities of the computer player character into simply several opponent characters for the human player to choose from, and it would already improve the game. Thus my programming has not been done in vain. The engine could be modified and the game would be easier to play.

This project helped me realize that I must not always rely merely on my own ideas and must always consult or at least read a lot about the design part first. I learned not to hurry to sit down and write the code – now I will always consider the design of the software first. As for the user interface, I did not get any bad reviews since everything was quite straightforward and the directions are given right during the game play in the toolbar panel below the game. There could always be given more directions in case there were not enough of them. The interface of the game is a pitch-black screen, and it could also be improved easily since the screen consists of the cells, and thus all that needs to be done is prettier images.

Overall it all seems to be the details, and I am generally satisfied with the knowledge I obtained as a programmer, and as a designer, although the design was not in

any case an important aspect of this project originally. And I am glad that I made mistakes along the way so I am more considerate in my character from now on.

References

[1] Free Pacman: http://www.freepacman.org/

[2] How to write a game design: http://www.pagtech.com/2006/07/04/how-to-write-a-game-design-doc/

[3] C++ Game Programming tutorial: http://www.cppgameprogramming.com/cgi/nav.cgi?page=index

[4] SDL tutorial:

http://www.sdltutorials.com/sdl-tutorial-basics/

[5] *Problem Solving with C++ - The Object of Programming* by Walter Savitch, Fifth Edition, Addison-Wesley Pub.

[6] *Beginning C++ Game Programming* by Michael Dawson, Second Edition, Cengage Learning Pub.

[7] *Creating Games in C++: A Step-by-Step Guide* by David Conger and Ron Little, New Riders Pub.

[8] *Focus on SDL* by Ernest Pazera, 2002 Edition, Premier Press Pub.

Images used:

[9] http://subscribe.ru/archive/rest.brain.labirints/200707/14181626.html/Ypitan.png

Appendix A

Timeline:

The interim and later the final reports were to be written throughout the allocated project time.

September Week 1: Come up with several ideas. Start calculating what I am capable of, so that I do not run into problems because of time limits.

September Week 2: Write out the ideas that most interest me on the cards, place the cards in front, and try to find free books and free tutorials on each of the ideas. The one that has more books and tutorials on it is a winner!

After choosing a supervisor and the topic “Game in C++”:

September Week 3: start reading the book *Problem Solving with C++* by Walter Savitch (because I have it at home). Go over it through the next week.

September Week 4: find more books and tutorials on game-making in C++. Find out what else I need to know about starting to learn C++ game programming.

October Week 1: Set up the environment: decide upon all the right software.

It turned out that I need to configure some library in order to create a game (for graphics, game controls, etc.). Then it turns out that a certain library is recommended for ordinary simple graphics, and is called Simple DirectMedia Library. Next thing I find is that my current compiler Dev-C++ is difficult to configure with the SDL library, so I decide to install the Visual Studio C++ 2008 Express Edition (because it´s also free). It took me a week to find all of this out, and finally everything is settled.

October Week 2: With everything set up, start getting ideas concerning game design while reading tutorials on both C++ and SDL.

October Week 3: Make a first suggestion for the game in terms of design. Continue reading tutorials on both C++ and SDL.

October Week 4: Draw a graphical draft of the game, show it to supervisor and make the idea about how the game will work clear to him.

November Week 1: The supervisor agreed to the game and informed me that I can continue with it.

November Week 2 – 4: program the game.

January Week 1: program the game.

After handing in Interim Report:

January Week 2-3: improve the code.

January Week 4: clean the unnecessary parts of code, write more comments.

February Week 1-2: see if the code can be improved. If so, improve it. Keep on doing so till the result is satisfactory.

February Week 3-4: start learning about how to test the code, and start applying different methods to test it.

March week 1: clean the unnecessary parts of code, write more comments.

March week 2: repeat the tests of the code, and make a table of some sort to record the data about those tests.

March week 3: check the final report for grammar errors.

Then make all of the code presentable and ready for final report.

March week 4: organize the final report, and finish it.

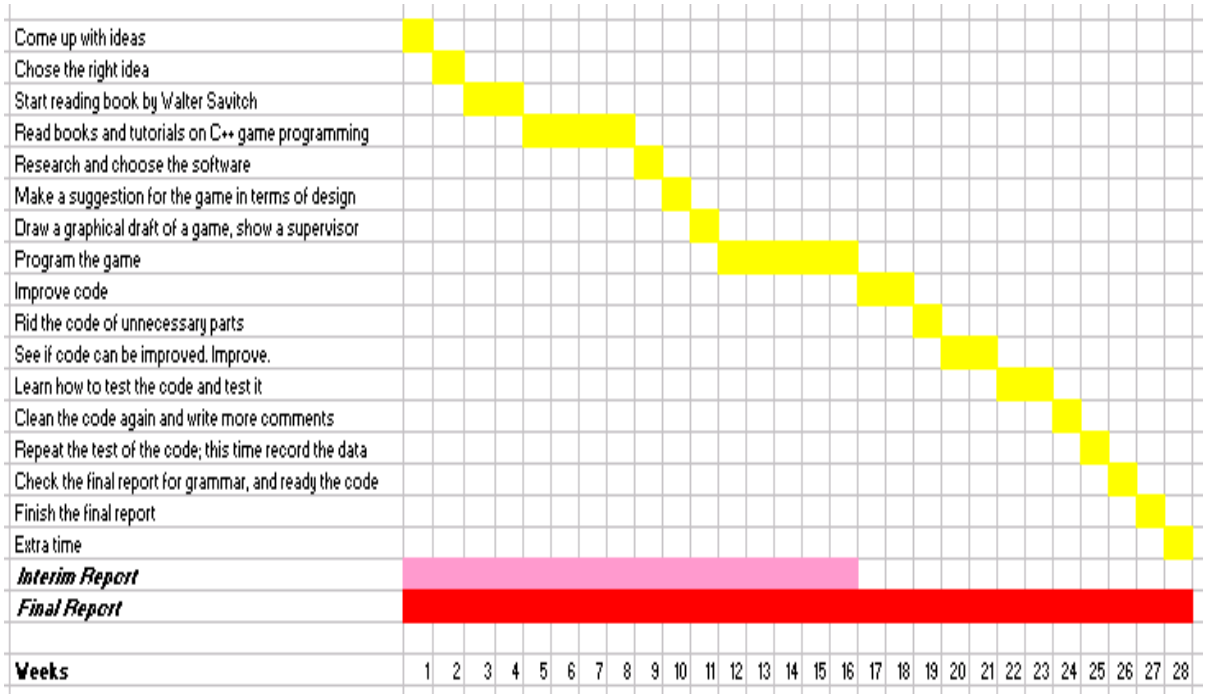
April Week 1: start preparing for presentations of the project. Use extra time in case an unexpected problem pops up.

April Week 2: Hand in the final report.

The above timeline was designed with certain risk factors in mind, and there is a day I would always reserve for the end of the week just in case.

Visualization of the timeline:

Figure 7 Time outline



Some changes took place and extra time was required to prepare for the presentation, although the material for the presentation was already gathered and written in the final report, i.e. the material was taken from almost finished then final report and used in the presentation.

More time was spent than expected setting up the Visual Studio C++ and managing it.

Less time than expected was spent on evaluation since it was being done at the same time of implementation mostly, to check the labyrinth, the characters moving, and other such technical details.

Appendix B

images.h

```
//declaring of the pictures (the future ones for now)
SDL_Surface *screen, *screen1;
SDL_Surface *back;
SDL_Surface *pers;
SDL_Surface *comp;
SDL_Surface *lose;
SDL_Surface *win;
SDL_Surface *path;
SDL_Surface *diamond;
SDL_Surface *bar_diamond;
SDL_Surface *clear_screen;
SDL_Surface *clear_vis;
SDL_Surface *squares[8];
SDL_Surface *menu;
SDL_Surface *rules;
SDL_Surface *menu_start;
SDL_Surface *menu_exit;
SDL_Surface *dice[7];

// the functions below are specifically written to make life easier
// while working with SDL. For instance, some of the additional
//complicated functions and operations that the SDL library offers,
//are ommitted.

//simply draw the picture
void img_draw ( SDL_Surface *img, int x, int y )
{
	SDL_Rect dest;
	dest.x = x;
	dest.y = y;
	SDL_BlitSurface ( img, NULL, screen, &dest );
}

//draws the painted over rectangle
void img_rect ( int x, int y, int w, int h, Uint8 R = 255, Uint8 G = 255, Uint8 B = 255 )
{
```

```
        Uint32 color = SDL_MapRGB ( screen->format, R, G, B );
        SDL_Rect dest;
        dest.x = x;
        dest.y = y;
        dest.w = w;
        dest.h = h;
        SDL_FillRect ( screen, &dest, color );
}

//this function draws the text to the screen (fonts are predifined).
//this function is the most time-consuming function of the game, and it
//of all the functions slows the game down.
void img_text ( const char* message, int size, int x, int y, Uint8 R = 0, Uint8 G = 0, Uint8
B = 0 )
{
        SDL_Color color = {R, G, B, 0};
        SDL_Rect dest;
        dest.x = x;
        dest.y = y;
        //this font file was found with great difficulty for both
        //english and cyrillic characters.
   //cyrillic characters are for my personal future use.
        TTF_Font *fnt = TTF_OpenFont ( "un1251n.ttf", size );
        SDL_Surface *sText = TTF_RenderText_Blended ( fnt, message, color );
        SDL_BlitSurface ( sText, NULL, screen, &dest );
        SDL_FreeSurface ( sText );
        TTF_CloseFont ( fnt );
}
```

engine.h

```
#include <time.h>

struct coor {
        int x, y;
};
coor cuser, ccomp, cdiamond;



//function to convert the int to a number
void itoa ( int i, char* s )
{
```

```
        if ( i < 0 )
                i = 0;

        int p = 0;

        char buf[10];

        do
        {
                buf[p] = i % 10 + 48;
                i /= 10;
                p++;
        }
        while ( i );

        for ( int a = 0;a < p;a++ )
                s[a] = buf[p-a-1];

        s[p] = 0;
}

//loss from cells
//unsigned short int square_damage[6] = {0, 10, 15, 1000, 40, 20};
//types of cells
//enum square_type {CLEAR, WALL1, WALL2, WALL3, MONSTER, TRAP};


//class that represnts a cell
class square
{
        public:
                bool way, path;
                char weight;
                bool diamond;
                square()
                {
                        weight = 0;
                        path = 0;
                        way = 0;
                        diamond = false;
                }

                //the cell gets drawn
                virtual void draw(int x, int y) const
                {
                        img_draw ( squares[0], x, y );
```

```
		}

		//the cell gets drawn
		void draw(SDL_Surface *sur, int x, int y)
		{
			img_draw ( sur, x, y );
		}

		//Is the cell an obstacle (that is wall)?
		virtual bool wall() const
		{
			return 0;
		}

};



class square_wall1: public square
{
	void draw(int x, int y) const{
		img_draw ( squares[1], x, y );
	}
	bool wall() const
	{
		return 1;
	}
};

class square_wall2: public square
{
	void draw(int x, int y) const{
		img_draw ( squares[2], x, y );
	}
	bool wall() const
	{
		return 1;
	}

};

class square_wall3: public square
{
	void draw(int x, int y) const{
		img_draw ( squares[3], x, y );
	}
```

```
        bool wall() const
        {
                return 1;
        }

};


class field
{

        private:

                //the labyrinth matrix
                square*** m;
                //auxiliary matrix for reference points/cells
                int* r[2];
                int h;
                //gives out a random reference point
                //(for the walls' generator)
                int getrandom ( int &x, int &y )
                {
                        int i;

                        if ( h )
                                i = rand() % h;
                        else
                                i = 0;

                        x = r[0][i];

                        y = r[1][i];

                        r[0][i] = r[0][h];

                        r[1][i] = r[1][h];

                        return h-- + 1;
                }

        public:
                int size_x, size_y; //the size of the labyrinth

                //int step = 32, startx = 0, starty = 0, done,
                //winner, mode = 1,
                //delay = 50, count_steps, FPS, iter;
```

```
		//int SCREEN_X = 800;
		//int SCREEN_Y = 600;



		//the drawing of the screen
		void view ( int Steps, int CountSteps, int CanSteps, int Ident, bool flag )
const
		{
			for ( int y = -starty; y <= ( int ) SCREEN_Y / step - starty; y++ )
			{
				for ( int x = -startx; x <= ( int ) SCREEN_X / step - startx;
x++ )
				{
	//we only draw within the defined size of the labyrinth.
	// this is basically for future use in case the
	// labyrinth won't be fitting into the defined size of a screen.
	//
					if ( x > size_x || y > size_y )
						continue;

					if ( !m[y][x]->wall() )
					  m[y][x]->draw ( ( x + startx ) *step, ( y + starty )
*step );

					if(m[y][x]->diamond) {
						if (y == size_y / 2 && x == size_x / 2)
							m[y][x]->draw ( squares[7], ( x +
startx ) *step, ( y + starty ) *step );

						m[y][x+1]->draw ( ( x + 1 ) *step, ( y )
*step );  //->
// | (down along the Y-axis towards the plus/positive)
						m[y+1][x]->draw ( ( x ) *step, ( y + 1) *step
);
						m[y][x-1]->draw ( ( x - 1 ) *step, ( y ) *step
); // <-
// | (up along the Y-axis towards the minus/negative)
						m[y-1][x]->draw ( ( x  ) *step, ( y - 1 ) *step
);

						m[y+1][x+1]->draw ( ( x + 1 ) *step, ( y +
1) *step );
						m[y+1][x-1]->draw ( ( x - 1 ) *step, ( y + 1)
*step );
```

```
                                        m[y-1][x-1]->draw ( ( x - 1 ) *step, ( y - 1)
*step );
                                        m[y-1][x+1]->draw ( ( x + 1 ) *step, ( y - 1)
*step );

                                }


                                if (
// ---- the horizontal wall above
                                    ( y == 0 && x >= 0 )||
// |    the vertical wall on the left
                                    ( y >= 0 && x == 0 ) ||
//  --- the horizontal wall below
                                    ( y == size_y && x >= 0 ) ||
//     | the vertical wall to the right
                                    ( y >= 0 && x == size_x )
                                  )

                                    m[y][x]->draw ( ( x + startx ) *step, ( y +
starty ) *step );

//draws the path
//if ( m[y][x]->path )
//img_draw ( path, ( x + startx ) *step, ( y + starty ) *step );

                                        }
                                }

                                //--drawing of the exit door --
                                //if (flag && Ident == 1) {
                            img_draw ( squares[6], ( size_x - 1 + startx ) *step, ( size_y - 1 +
starty ) *step );
                                  //SDL_Flip ( screen );
                                //}

                                //the toolbar panel
                                img_rect ( 0, SCREEN_Y - 55, SCREEN_X, SCREEN_Y );
                                char steps_text[10];
                                itoa ( CountSteps, steps_text );
                                img_text ( "Steps:", 22, 5, SCREEN_Y - 39 );
                                img_text ( steps_text, 22, 80, SCREEN_Y - 39 );

                                img_text ( "Steps remaining:", 22, 135, SCREEN_Y - 39 );
                                itoa ( Steps, steps_text );
                                img_text ( steps_text, 22, 330, SCREEN_Y - 39 );
```

```
		//the die
		img_draw(dice[CanSteps], 755, 549);

		//the diamond
		if (flag && Ident == 1)
			img_draw ( bar_diamond, 705, SCREEN_Y - 47);

}

field() {}

//the labyrinth generation
void init ( int new_size_x, int new_size_y )
{
		srand ( ( unsigned ) time ( NULL ) * rand() % 100000 );
		//srand ( 1 );
		int i, j = 0, x, y, sx, sy;

		size_x = new_size_x;

		size_y = new_size_y;

		if ( size_x < 10 )
			size_x = 10;

		if ( size_y < 10 )
			size_y = 10;

		if ( size_x % 2 )
			size_x++;

		if ( size_y % 2 )
			size_y++;

		// +1, since there is also the wall made of bricks
		//that goes across the whole labyrinth.
		m = new square** [size_y+1];

		for ( i = 0; i < size_y + 1; i++ )
		{
			m[i] = new square* [size_x+1];

			for ( j = 0; j < size_x + 1; j++ )
				m[i][j] = new square;
		}
```

```
                    //-- "-1", for making it even --
                    r[0] = new int[ ( size_x/2-1 ) * ( size_y/2-1 ) ];
                    r[1] = new int[ ( size_x/2-1 ) * ( size_y/2-1 ) ];
                    short direction[8] = { -1, 0, 1, 0, 0, 1, 0, -1};

                    //--filling in file by brick labyrinth wall
                    //above and below along the X-axis.
                    //
                    for ( i = 0; i <= size_x; i++ )
                    {
                            delete m[0][i];
                            m[0][i]=new square_wall3;
                            delete m[size_y][i];
                            m[size_y][i]=new square_wall3;
                    }
                    //--

                    //--filling in file by brick labyrinth wall
                   //on the right and on the left along the Y-axis.
                    //
                    for ( i = 0; i <= size_y; i++ )
                    {
                            delete m[i][0];
                            m[i][0]=new square_wall3;
                            delete m[i][size_x];
                            m[i][size_x]=new square_wall3;
                    }
                    //--

                    j = 0;
            // "y = 2"
            // "size_y - 1"
            // two of the cells on the field are not to be touched,
    //since they are left for the two players. These cells are
    //situated vertically to each other, like so:
                    //   -------------------
                    //  |H******************|
                    //  |******************|
                    //  |******************|
                    //  |******************|
                    //  |******************|
                    //  |C******************|
                    //   -------------------
// Above H stands for human player, and C - for computer player.
```

```
for ( y = 2; y < size_y - 1; y += 2 )
{
        for ( x = 2; x < size_x - 1; x += 2 )
        {
                r[0][j] = x;
                r[1][j] = y;
                j++;
        }
}

h = j - 1; //the number of cells (walls)
          //there were generated in labyrinth

//Creating of the walls in the labyrinth:
// firstly we get a random reference point/cell
while ( getrandom ( sx, sy ) )
{
        if ( m[sy][sx]->wall() )
                continue;

        x = 0;
        y = 0;

        j = rand() % 4;

        x = direction[j];
        y = direction[j+4];

        j = 0;

        int random_wall = rand() % 2;

//as long as the way/path is clear, meaning there is no wall there
        while ( !m[sy][sx]->wall() )
        {
                if ( j > 20 )
                {
                        delete m[sy][sx];
                        switch(random_wall)
                        {
                                case 0:
                                        m[sy][sx] = new
square_wall1;
                                        break;
                                case 1:
```

```
                                        m[sy][sx] = new
square_wall2;
                                        break;
                                }
                                break;
                        }

                        delete m[sy][sx];
                        switch(random_wall)
                        {
                                case 0:
                                        m[sy][sx] = new square_wall1;
                                        break;
                                case 1:
                                        m[sy][sx] = new square_wall2;
                                        break;
                        }

                        sx += x;
                        sy += y;
                        delete m[sy][sx];
                        switch(random_wall)
                        {
                                case 0:
                                        m[sy][sx] = new square_wall1;
                                        break;
                                case 1:
                                        m[sy][sx] = new square_wall2;
                                        break;
                        }
                        sx += x;
                        sy += y;
                        j++;
                }
        }
        //-- building the diamond into the labyrinth --
        delete m[size_y / 2][size_x / 2];
        m[size_y / 2][size_x / 2] = new square;
        m[size_y / 2][size_x / 2]->diamond = true;
        cdiamond.x = size_x / 2;
        cdiamond.y = size_y / 2;

}

~field()
{
```

```
 //Additional Feature:
 //The exit door gets deleted while you are not in poccession of a  //diamond
 //or haven't reached it yet. The door signifies a portal that is only //open for you
 //while you are holding the diamond. This additional feature was //proposed
 //by a friend, and it is more of a logical meaning.
 //Implement it by deleting the comment tags /*  */

                        /*int i;
                        delete[] r[0];
                        delete[] r[1];

                        for ( i = 0; i < size_y + 1; i++ ){
                                for ( int j = 0; j < size_y + 1; j++ )
                                        delete m[i][j];
                                delete[]m[i];
                        }

                        delete[]m;

                        for ( i = 0; i < size_y + 1; i++ )
                                delete[]a[i];

                        delete[]a;/**/
                }

                friend class player;
};

class player:public field
{
   public:
                int st;
                field* l;
                int x, y;
                int numb_moves; // the number of steps remaining //(on the die)
                int count_moves; //the number of moves the player can make //at once.
                bool status;            // the status of the current move
                //whose turn is it, the user's or the computer's?
                //(1- for user, 0- for the computer)
                int ident;
                int count_steps;        //the number of all the already made //moves/steps
                bool flag;                      // do we hold the diamond at the //moment?


                //--throwing the die --
                void cubic() {
```

```
        int i = rand() % 6;
        switch (i) {
            case 0: i++;
                    img_draw(dice[1], 755, 549); break;
            case 1: img_draw(dice[1], 755, 549); break;
            case 2: img_draw(dice[2], 755, 549); break;
            case 3: img_draw(dice[3], 755, 549); break;
            case 4: img_draw(dice[4], 755, 549); break;
            case 5: img_draw(dice[5], 755, 549); break;
            case 6: img_draw(dice[6], 755, 549); break;
        }
        numb_moves = i;  // initialising the counter of
                         //runs/steps of the player
        count_moves = i;
        SDL_Flip ( screen );
}


player ( field* f )
{
        l = f;                  // initialising the labyrinth
        count_steps = 0;
        flag = false;
}

void SetCoor() {
        //annuling (restoring to zero) the counter of the //turns
        //of the player.
        numb_moves = 0;
        count_moves = 0;
        if (ident == 1) {   //user
          x = 1, y = 1;
          cuser.x = 1, cuser.y = 1;
        }
        else if (ident == 0){           // computer
          x = 1, y = 15;
          ccomp.x = 1, ccomp.y = 15;
        }
}

        //Important Feature:
//calculates the shortest path from point A (where you are) to
//point B (the cell you point over your mouse and click). This
//is called a cheat.

        void go_ ( short x, short y, int gox, int goy, int steps = 0 )
```

```
{
        int i, j;
//      l->a[x][y] = 1;
        l->m[x][y]->way = 1;

        if ( st > steps && x == gox && y == goy )
        {
                for ( i = 0; i < l->size_y; i++ )
                        for ( j = 0; j < l->size_x; j++ )
                                l->m[i][j]->path = l->m[i][j]->way;

                st = steps;
        }

        if ( steps < st )
        {
                if ( !l->m[x][y+1]->way && !l->m[x][y+1]->wall() )
                        go_ ( x, y + 1, gox, goy, steps + 1 );

                if ( !l->m[x+1][y]->way && !l->m[x+1][y]->wall() )
                        go_ ( x + 1, y, gox, goy, steps + 1 );

                if ( !l->m[x][y-1]->way && !l->m[x][y-1]->wall() )
                        go_ ( x, y - 1, gox, goy, steps + 1 );

                if ( !l->m[x-1][y]->way && !l->m[x-1][y]->wall() )
                        go_ ( x - 1, y, gox, goy, steps + 1 );
        }

        l->m[x][y]->way = 0;

//      l->a[x][y] = 0;
}


//the mouse control
void go ( int goy, int gox )
{

                st = 100;
                go_ ( y, x, gox, goy );
                int i = 1;

                if ( st != 100 )
                {
                        //--movinf towards the specified cell
```

```
//with number of moves taken into account (numb_moves)--
                //cannot make more steps than you have on a die.
                                        while ( (( x != goy ) || (y != gox)) && (
numb_moves > 0 ) )
                                        {
                                                l->m[y][x]->path = 0;

                                                if ( l->m[y][x+1]->path )
                                                        x++;
                                                else
                                                        if ( l->m[y+1][x]->path )
                                                                y++;
                                                        else
                                                                if ( l->m[y-1][x]->path )
                                                                        y--;
                                                                else
                                                                        if ( l->m[y][x-1]-
>path )
                                                                                x--;

                                                draw();

                                                SDL_Delay ( 50 );

                                                SDL_Flip ( screen );

                                                FPS++;

                                                //number of steps
                                                count_steps++;

                                                numb_moves--; // diminish the
//counter of made amount of runs/steps by the player

                                        }

                                        /*if ( ident == 0) { //if this was a computer's move
there's no need to show its path to the user!
                                                int sx = x;
                                                int sy = y;
                                                while ( (( sx != goy ) || (sy != gox)) )
                                                {
                                                        l->m[sy][sx]->path = 0;

                                                        if ( l->m[sy][sx+1]->path )
                                                                sx++;
```

```
                                                    else
                                                            if ( l->m[sy+1][sx]->path )
                                                                    sy++;
                                                            else
                                                                    if ( l->m[sy-1][sx]-
>path )
                                                                            sy--;
                                                                    else
                                                                            if ( l-
>m[sy][sx-1]->path )
                                                                                    sx--;
                                                    }
                                                    l->m[sy][sx]->path = 0;
                                            }*/

                                            l->m[y][x]->path = 0;
                                    }
                            }
                            void PressAnyKey() {
                                    SDL_Event event;
                                    bool done = false;
                                    while (!done) {
                                            while (SDL_PollEvent ( &event )) {
                                                    //-- the exit happened --
                                                    if ( event.type == SDL_QUIT )
                                                    {
                                                            done = 1;
                                                            close_game = 0;
                                                            break;
                                                    }

                                                    if ( event.type == SDL_KEYDOWN )
                                                    {
                            //Additional

//--if ESC is pressed down - the quit happens
                                                            if ( event.key.keysym.sym ==
SDLK_ESCAPE )
                                                            {
                                                                    done = 1;
                                                                    break;
                                                            }
                                                    }
                                                    //the event that is generated when any
//key on keyboard is pressed
                                                    if ( event.type == SDL_KEYDOWN ) {
```

```
                                        done = true;
                                        break;
                                }
                        }
                }
        }

        void ClearOldCoorDiamond(int x, int y) {
//l->m[y][x]->draw ( back, ( x + startx ) *step, ( y + starty ) *step);

                l->m[y][x+1]->draw ( back, ( x + startx + 1 ) *step, ( y + starty)
*step );  //->
                l->m[y+1][x]->draw ( back, ( x + startx ) *step, ( y + starty + 1)
*step );  // | (down the Y-axis towards the //plus/positive)
                l->m[y][x-1]->draw ( back, ( x + startx - 1 ) *step, ( y + starty )
*step ); // <-
                l->m[y-1][x]->draw ( back, ( x + startx ) *step, ( y + starty - 1 )
*step ); // | (up the Y-axis towards the minus/negative)

                l->m[y+1][x+1]->draw ( back, ( x + 1 ) *step, ( y + 1) *step );
                l->m[y+1][x-1]->draw ( back, ( x - 1 ) *step, ( y + 1) *step );
                l->m[y-1][x-1]->draw ( back, ( x - 1 ) *step, ( y - 1) *step );
                l->m[y-1][x+1]->draw ( back, ( x + 1 ) *step, ( y - 1) *step );

        }

        //the drawing of the character
        void draw()
        {

        //--making the drawing of the are for the given player --
                if (ident == 1) { // user
    //-- drawing the character in given/specified coordinates --
                        if ( flag ) {
                                if ( cuser.x == cdiamond.x && cuser.y ==
cdiamond.y )  {
                                        ClearOldCoorDiamond(cdiamond.x,
cdiamond.y);
                                        l->m[cdiamond.y][cdiamond.x]->diamond =
false;
                                        l->m[y][x]->diamond = true;
                                        cdiamond.x = x;
                                        cdiamond.y = y;
                                }
                        }
                        if ( x == cdiamond.x && y == cdiamond.y )  {
```

```
                        flag = true;
                    }
                    else if ( x != cdiamond.x || y != cdiamond.y )  {
                        flag = false;
                    }
                    cuser.x = x;
                    cuser.y = y;
                    l->view (numb_moves, count_steps, count_moves, ident,
flag);
                    img_draw ( pers, ( x + startx ) *step, ( y + starty ) *step );
                    img_draw ( comp, ( ccomp.x ) *step, ( ccomp.y ) *step );
                }
                else if (ident == 0){ // same for computer
                    if ( flag ) {
                        if ( ccomp.x == cdiamond.x && ccomp.y ==
cdiamond.y ) {
                            ClearOldCoorDiamond(cdiamond.x,
cdiamond.y);
                            l->m[cdiamond.y][cdiamond.x]->diamond =
false;
                            l->m[y][x]->diamond = true;
                            cdiamond.x = x;
                            cdiamond.y = y;
                        }
                    }
                    if ( x == cdiamond.x && y == cdiamond.y ) {
                        flag = true;
                    }
                    else if ( x != cdiamond.x || y != cdiamond.y )  {
                        flag = false;
                    }
                    ccomp.x = x;
                    ccomp.y = y;
                    l->view (numb_moves, count_steps, count_moves, ident,
flag);
                    img_draw ( comp, ( x + startx ) *step, ( y + starty ) *step );
                    if ( cuser.x == 0 && cuser.y == 0)
                        img_draw ( pers, step, step );
                    else
                        img_draw ( pers, ( cuser.x ) *step, ( cuser.y ) *step
);
                }

                delete l->m[y][x];
                l->m[y][x] = new square;
```

```
                        if (ident) {
                           //and the winner is...
                                if (flag)
                                if ( x == l->size_x - 1 && y == l->size_y - 1 )
                                {
                                        img_rect ( SCREEN_X / 2 - 250, SCREEN_Y / 2 -
45, 500, 95 );
                                        img_text ( "You have won!", 50, SCREEN_X / 2 -
150, SCREEN_Y / 2 - 25 );
                                        img_draw ( win, SCREEN_X / 2 - 250,
SCREEN_Y / 2 - 45 );
                                        winner = 1;
                                        img_text ( "Press any button...", 15, 355,
SCREEN_Y - 34 );
                                        SDL_Flip ( screen );
                                        PressAnyKey();
                                        SDL_Delay(100);

                                }
                        }
                        else {
                                if(flag)
                                if ( x == l->size_x - 1 && y == l->size_y - 1 )
                                {
                                        img_rect ( SCREEN_X / 2 - 250, SCREEN_Y / 2 -
45, 500, 95 );
                                        img_draw ( lose, SCREEN_X / 2 - 250,
SCREEN_Y / 2 - 45 );
                                        img_text ( "You have lost!", 50, SCREEN_X / 2 -
150, SCREEN_Y / 2 - 25 );
                                        img_text ( "Press any button...", 15, 355,
SCREEN_Y - 34 );
                                        winner = 1;
                                        Lose = 1;
                                        SDL_Flip ( screen );
                                        PressAnyKey();
                                        SDL_Delay(100);
                                }
                        }
                }
                //the player makes a move
                bool StepPlayer ( int xs, int ys )
                {
//--as long as the player makes another step, move
//the player's character to the
//specified (by the player) cell of the labyrinth.
```

```
                if(numb_moves > 0) {
            //--if this is not a wall, then the step can be taken--
                    if ( !l->m[y+ys][x+xs]->wall() )
                    {
                    x += xs;
                    y += ys;

                                        //responds to the automatic
//scrolling during the move of the character
                                        if ( grid_x < l->size_x )
                                        {
                                                startx = grid_x / 2 - x;
                                                if ( startx > 0 )
                                                        startx = 0;
                                                if ( startx < grid_x - l->size_x )
                                                        startx = grid_x - l->size_x -
1;
                                        }
                                        if ( grid_y < l->size_y )
                                        {
                                                starty = grid_y / 2 - y;
                                                if ( starty > 0 )
                                                        starty = 0;
                                                if ( starty < grid_y - l->size_y )
                                                        starty = grid_y - l->size_y -
1;
                                        }

                                        //number of steps
                                        count_steps++;

                                        numb_moves--; //decreasing the
//counter by 1 of the player's made move/step

                    }

                }
                return 1;
            }

};
```

OP_Maze_RP.cpp

```
// OP_Maze_RP.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"


#ifdef WIN32
#pragma comment (lib, "SDL.lib")
#pragma comment (lib, "SDLmain.lib")
#pragma comment (lib, "SDL_image.lib")
#pragma comment (lib, "SDL_ttf.lib")
#pragma comment (lib, "SDL_mixer.lib")
#endif
#include "SDL.h"
#include "SDL_image.h"
#include "SDL_mixer.h"
#include "SDL_ttf.h"
#include <stdio.h>
#include <stdlib.h>

int SCREEN_X = 800;
int SCREEN_Y = 600;
int step = 32, startx = 0, starty = 0, done, winner, Lose, mode = 1, delay = 50, FPS, iter;
bool monsters = 0;
int grid_x = SCREEN_X / step;
int grid_y = SCREEN_Y / step;
int close_game = 0;

#include "images.h"  // the functions of making life easier while //working with SDL
#include "engine.h"  // engine of the game, the players' classes, //graphics, etc.

//function for calculating fps (frames per second)
void* my_callback_param;

Uint32 my_callbackfunc ( Uint32 interval, void *param )
{
        printf ( "FPS: %d\n", FPS );
        FPS = 0;
        return interval;
}

int Menu() {

        img_draw ( menu, 0, 0 );
```

```
SDL_Flip ( screen );

SDL_Event event;
SDL_ShowCursor ( 1 );
bool done = 0;
while(!done) {
  while ( SDL_PollEvent ( &event ) ) {
                //--the exit happened here --
                if ( event.type == SDL_QUIT )
                {
                        return false;
                }

                if ( event.type == SDL_KEYDOWN )
                {
                        //--when ESC is pressed down, the exit happens.
                        if ( event.key.keysym.sym == SDLK_ESCAPE )
                        {
                                return false;
                        }
                }
                if ( event.type == SDL_MOUSEBUTTONDOWN ) {
                        if ( (event.button.y > 201 && event.button.y < 341) &&
                                (event.button.x > 20 && event.button.x < 180) ) {
                                        img_draw ( rules, 0, 0 );
                                        SDL_Flip ( screen );
                        }
                        if ( (event.button.y > 191 && event.button.y < 368) &&
                                (event.button.x > 285 && event.button.x < 501) ) {
                                        img_draw ( menu_start, 0, 0 );
                                        SDL_Flip ( screen );
                        }
                        if ( (event.button.y > 453 && event.button.y < 589) &&
                                (event.button.x > 471 && event.button.x < 624) ) {
                                        img_draw ( menu_exit, 0, 0 );
                                        SDL_Flip ( screen );
                        }
                }
                if ( event.type == SDL_MOUSEBUTTONUP ){
                                printf("Coor X: %d\n", event.button.x);
                                printf("Coor Y: %d\n", event.button.y);


                        img_draw ( menu, 0, 0 );
                        if ( (event.button.y > 201 && event.button.y < 341) &&
```

```
                                        (event.button.x > 20 && event.button.x < 180) ) {
                                          img_text ( "Each turn roll a die,", 22, 580, 10,
255, 50, 40 );
                                                img_text ( "and using arrows", 22, 580, 30,
255, 50, 40 );
                    img_text ( "try making it to", 22, 580, 50, 255, 50, 40 );
                    img_text ( "the diamond and", 22, 580, 70, 255, 50, 40 );
                    img_text ( "out the portal door.", 22, 580, 90, 255, 50, 40 );
                                                img_text ( "Without the diamond", 22, 580,
110, 255, 50, 40 );
                                                img_text ( "one cannot win.", 22, 580, 130,
255, 50, 40 );
                                                img_text ( "To steal the diamond", 22, 580,
150, 255, 50, 40 );
                                                img_text ( "from the opponent,", 22, 580,
170, 255, 50, 40 );
                                                img_text ( "catch him and", 22, 580, 190,
255, 50, 40 );
                                                img_text ( "outstrip him.", 22, 580, 210,
255, 50, 40 );
                                                img_text ( "Good luck!", 22, 580, 230, 255,
50, 40 );
                                    }
                                    if ( (event.button.y > 191 && event.button.y < 368) &&
                                            (event.button.x > 285 && event.button.x < 501) ) {
                                                return 1;
                                    }
                                    if ( (event.button.y > 453 && event.button.y < 589) &&
                                            (event.button.x > 471 && event.button.x < 624) ) {
                                              return 0;
                                    }
                                    SDL_Flip ( screen );
                            }

                  }
                }
                return 0;
}

//-- The function for the waiting of the user's (human player) next //throwing of a die--
//-- (it returns false in case the game has been aborted --
bool ThrowCub(player *user) {
  SDL_Event event;
  bool done = 0;
  img_text ( "Awaiting your turn...", 15, 415, SCREEN_Y - 34 );
  SDL_Flip ( screen );
```

```
  SDL_ShowCursor ( 1 );
  while(!done) {
          while ( SDL_PollEvent ( &event ) ) {
                  //--The game was aborted --
                  if ( event.type == SDL_QUIT )
                  {
                          return false;
                  }
                if ( event.type == SDL_MOUSEBUTTONUP && event.button.button
== SDL_BUTTON_LEFT ) {
                          if ((event.button.x >= 755 && event.button.x <= 794) &&
                                  (event.button.y >= 549 && event.button.y <= 588)
) {
                                  user->cubic();
                                  done = true;
                                  break;
                          }
                  }
          }
  }
  //-- Clearing of the "Awaiting your turn" writing --
  user->draw();
//user->view(user->numb_moves, user->count_steps,
//user->count_moves, 1, user->flag);
  SDL_Flip ( screen );
  return true;
}

//-- the function of the human player turn/move --
int ProgressUserPlayer(player &user, field &labirint) {

        //--
        //:the function returns the messages of the sdl types:
        //-1: the game was closed
        //1: the player has won
        //0: the player has finished his/her turn/move(s)
        //--
        user.status = true;
        user.ident = 1;

        //-throwing the die --
        if(!ThrowCub(&user)) return -1; //while the die was thrown, it
//was fixed that the game was aborted.

        bool done = 0;
        while(!done) {
```

```
		SDL_Event event;
	//--now it's the human player's turn :) --
	while ( SDL_PollEvent ( &event ) )
	{
		//-- the fixing --
		//user.numb_moves = 6;
		//--
		//--The human player only makes as many moves as he/she
//sees on the die :) -
		if ( user.numb_moves > 0 ) {
			//--The exit happened--
			if ( event.type == SDL_QUIT )
			{
				return -1;
			}

			//things that happen when the keyboard button is //pressed
			if ( event.type == SDL_KEYDOWN )
			{
				if ( winner )
				{
					return 1; //returns the result of the
//winning process (who wins)
				}
				SDL_ShowCursor ( 0 );

				//--when ESC is pressed - the exit happens
				if ( event.key.keysym.sym == SDLK_ESCAPE )
				{
					return -1;
				}
				else
					switch ( event.key.keysym.sym )
					{
			//the finding of the solution of the labyrinth
						case SDLK_HOME:
//--Sending the user to the exit of the labyrinth --
//--who will be walking to the particular/predefined cell
//-- and will be notified if he/she won
//by showing the picture of the winner.
//-- This will be done by comparing the coordinates of
//the coordinates of two players to
//-- the coordinate of the predifined cell of the exit door.
							user.go ( 23, 15 );
							if(winner)
```

```
//--this variable will hold the state of a "win"
                                                        continue;
                                                    break;
                                        }

        // using arrows to move the character (piece on the game "board")
                                    while ( event.type != SDL_KEYUP && !winner )
                                    {
        //the arrows up, down, left and right are to be used
                                        if ( event.key.keysym.sym == SDLK_UP )
                                            user.StepPlayer ( 0, -1 );
                                        if ( event.key.keysym.sym == SDLK_DOWN )
                                            user.StepPlayer ( 0, 1 );
                                        if ( event.key.keysym.sym == SDLK_LEFT )
                                            user.StepPlayer ( -1, 0 );
                                        if ( event.key.keysym.sym == SDLK_RIGHT )
                                            user.StepPlayer ( 1, 0 );

//--draw the character in a new position, that is as it moves --
                                        user.draw();
//--after re-drawing the state of the game is
//checked (win or lose that is)

                                        //--renew the changes on screen --
                                        SDL_Flip ( screen );
                                        SDL_Delay ( 100 );

                                        //-- claim to recieve the event/response
                                        SDL_PollEvent ( &event );

                                        if ( (user.numb_moves <= 0) && !winner) {
                                            return 0;
                                        }
                                        else if ( winner ){
                                            return 1;
                                        }

                    //--leaving the keyboard use and moving on to mouse events
                                        if ( event.type == SDL_MOUSEMOTION )
                                            break;
                                    }
                                    FPS++;

                                }//event generated as the mouse button gets pressed
                                else
// the click of the button was fixated, meaning the event was caught
```

```
                {
                                        if ( event.type ==
SDL_MOUSEBUTTONUP && winner )
                                        {
                                                return 1;
// the button is released (no longer pressed)
//and the winning is fixated
                                        }

                                        //the mouse controls
                                        if ( event.type ==
SDL_MOUSEBUTTONUP && event.button.button == SDL_BUTTON_LEFT &&
!winner ){
                                                printf("Coor X: %d\n",
event.button.x);
                                                printf("Coor Y: %d\n",
event.button.y);
//-- in case the click has happened in the panel under the labyrinth,
// the situation of the toolbar is taken into account
                                                if ( event.button.y <
SCREEN_Y - 55) {
                        //move the player to the given position
                                                        user.go (
event.button.x / step - startx, event.button.y / step - starty );
                        //cuser.x = event.button.x / step - startx;
                //cuser.y = event.button.y / step - starty;
                                                }
                                        }
                                        if ( (user.numb_moves <= 0) && !winner) {
                                                return 0;
                                        }
                                        else if ( winner ){
                                                return 1;
                                        }
                        }// the mouse click is fixated

                        if ( event.type == SDL_MOUSEMOTION && !winner )
                        {
                                //scrolling
                                SDL_ShowCursor ( 1 );

                                bool t=0;

                                if ( event.motion.x > ( SCREEN_X - 20 ) &&
SCREEN_X <= ( labirint.size_x + startx )*step )
                                {
```

```
                                        t = 1;
                                        startx--;
                                }

                                if ( event.motion.x < 20 && startx < 0 )
                                {
                                        t = 1;
                                        startx++;
                                }

                                if ( event.motion.y < 20 && starty < 0 )
                                {
                                        t = 1;
                                        starty++;
                                }

                                if ( event.motion.y > ( SCREEN_Y - 20 ) &&
SCREEN_Y <= ( labirint.size_y + starty )*step +24 )
                                {
                                        t = 1;
                                        starty--;
                                }

                                if(t)
                                {
                                        user.draw();
                                        SDL_Flip ( screen );
                                }

                                FPS++;
                        }//scrolling of the mouse
                        SDL_Delay ( 30 );
                }
//--making exactly as many steps as it is fixated on the die
                else {//-- the steps are all taken/finished up --
                        done = 1;
                        break;
                }
        }
        }
        user.status = false;
        return 0; // the player took his/her steps (made his/her move)
}

int ProgressCompPlayer(player &computer) {
        //--status of the game is now: computer's turn now --
```

```
            computer.status = true;
            computer.ident = 0;
            //-- throwing of a die --
            computer.cubic();
//-- calculate/decide upon a shortest path
//and move along it towards the exit door
            //--in view of the number of steps that were generated using a die
            if (computer.flag)
               computer.go(23, 15);
            else
                        computer.go(cdiamond.x, cdiamond.y);
            //ccomp.x = computer.x;
            //ccomp.y = computer.y;

            if (winner) {
               Lose = 1; //the user has lost the game
//--the computer player has made its move and its status now is:
        // "awaiting the human player's turn".
               computer.status = false;
               return 1;
            }
            else
                        return 0;
            //-- the computer has finished its move, waiting for its turn
            computer.status = false;
}

int _tmain(int argc, _TCHAR* argv[])
{
            if ( SDL_Init ( SDL_INIT_TIMER | SDL_INIT_VIDEO ) < 0 )
                        exit ( 1 );
            atexit ( SDL_Quit );
            TTF_Init();
            atexit ( TTF_Quit );

            //the video regime
/*screen =
SDL_SetVideoMode(SCREEN_X,SCREEN_Y,32,SDL_HWSURFACE|SDL_DOUBLE
BUF|SDL_FULLSCREEN);  */
            screen = SDL_SetVideoMode ( SCREEN_X, SCREEN_Y, 32,
SDL_HWSURFACE | SDL_RESIZABLE | SDL_DOUBLEBUF);

            //the name of the window
            SDL_WM_SetCaption ( "Maze", "Maze" );

            #ifdef SOUND_ON
```

```
                if ( Mix_OpenAudio(44000, AUDIO_S16SYS, 2, 1024) < 0 ) {
                        fprintf(stderr,
                        "Warning: Couldn't set audio\n- Reason: %s\n",
                                SDL_GetError());
                }
        #endif

//converting of the pictures to boost speed of
//the appearance of the graphics
        pers = SDL_DisplayFormat ( IMG_Load ( "Gfx/pers2.png" ) );
        comp = SDL_DisplayFormat ( IMG_Load ( "Gfx/comp.png" ) );
        back = SDL_DisplayFormat ( IMG_Load ( "Gfx/clear.png" ) );
        lose = SDL_DisplayFormat ( IMG_Load ( "Gfx/lose.png" ) );
        win = SDL_DisplayFormat ( IMG_Load ( "Gfx/win.png" ) );
        path = SDL_DisplayFormatAlpha ( IMG_Load ( "Gfx/path.png" ) );
        diamond = SDL_DisplayFormatAlpha ( IMG_Load ( "Gfx/diamond.png" ) );
        bar_diamond = SDL_DisplayFormatAlpha ( IMG_Load (
"Gfx/toolbar_diamond.png" ) );
        clear_screen = SDL_DisplayFormatAlpha ( IMG_Load ( "Gfx/clear_screen.png"
) );
        clear_vis = SDL_DisplayFormatAlpha ( IMG_Load ( "Gfx/clear_vis.png" ) );
        menu =  SDL_DisplayFormatAlpha ( IMG_Load ( "Gfx/menu.png" ) );
        rules = SDL_DisplayFormatAlpha ( IMG_Load ( "Gfx/menu_rules.png" ) );
        menu_start = SDL_DisplayFormatAlpha ( IMG_Load ( "Gfx/menu_start.png" ) );
        menu_exit = SDL_DisplayFormatAlpha ( IMG_Load ( "Gfx/menu_exit.png" ) );

        squares[0] = SDL_DisplayFormat ( IMG_Load ( "Gfx/clear.png" ) );
        squares[1] = SDL_DisplayFormat ( IMG_Load ( "Gfx/wall1.png" ) );
        squares[2] = SDL_DisplayFormat ( IMG_Load ( "Gfx/wall2.png" ) );
        squares[3] = SDL_DisplayFormat ( IMG_Load ( "Gfx/wall3.png" ) );
        squares[4] = SDL_DisplayFormat ( IMG_Load ( "Gfx/monster2.png" ) );
        squares[5] = SDL_DisplayFormat ( IMG_Load ( "Gfx/trap.png" ) );
        squares[6] = SDL_DisplayFormat ( IMG_Load ( "Gfx/exit.png" ) );
        squares[7] = SDL_DisplayFormat ( IMG_Load ( "Gfx/diamond.png" ) );

        dice[0] = SDL_DisplayFormat ( IMG_Load ( "Gfx/cubic/0.png" ) );
        dice[1] = SDL_DisplayFormat ( IMG_Load ( "Gfx/cubic/1.png" ) );
        dice[2] = SDL_DisplayFormat ( IMG_Load ( "Gfx/cubic/2.png" ) );
        dice[3] = SDL_DisplayFormat ( IMG_Load ( "Gfx/cubic/3.png" ) );
        dice[4] = SDL_DisplayFormat ( IMG_Load ( "Gfx/cubic/4.png" ) );
        dice[5] = SDL_DisplayFormat ( IMG_Load ( "Gfx/cubic/5.png" ) );
        dice[6] = SDL_DisplayFormat ( IMG_Load ( "Gfx/cubic/6.png" ) );


        if ( screen == NULL )
                exit ( 1 );
```

```
//send to separate flow function for
//calculation of FPS (frames per second)
        SDL_TimerID my_timer_id = SDL_AddTimer ( 1000, my_callbackfunc,
my_callback_param );

//--while the window with the game is not closed,
//the game is continuing --
        while ( close_game == 0 )
        {
                if (!Menu() )
                        return 0;

                //--annuling/restoring the initial data --
                startx = starty = winner = Lose = done = 0;

                //--turning of the cursor--
                SDL_ShowCursor ( SDL_DISABLE );

//--declaring the object of the class "field"
//(the area of the labyrinth) --
                field labirint;

//--now building the labyrinth the size of which
//we want to be 23 cells along the X-axis
                //-- and 16 cells along the Y-axis --
                labirint.init ( 24, 16 );

                //-- creating two players (human and computer players)--
                //--::user (human player)= true;
                //--::computer player= false;
                //--drawing both characters
                img_draw(clear_screen, 0, 0);

                player computer ( &labirint );
                computer.ident = 0;
                computer.SetCoor();
                computer.draw();

                player user ( &labirint );
                user.ident = 1;
                user.SetCoor();
                user.draw();
```

```
                //-- applying the changes at redrawing --
                SDL_Flip ( screen );

                while ( done == 0 )
                {

                        user.count_moves = 0;
                        if (computer.flag)
                          user.view(0, user.count_steps, user.count_moves, 1, false);
                        else if ( user.flag )
                          user.view(0, user.count_steps, user.count_moves, 1, true);

                        if ( !user.flag && !computer.flag )
                                user.view(0, user.count_steps, user.count_moves, 1, false);


                        //-- Now's the time to make a turn/move! --
                        switch ( ProgressUserPlayer(user, labirint) ) {
                                //--the exit was fixated --
                                case -1: done = 1;
                                                break;
                                //--the user has won, start new game --
                                case 1: done = 1;
                                                break;
//-- the user made a move, now awaiting the computer player's turn --
                                case 0:
                                                switch(ProgressCompPlayer(computer)) {
                //--the computer player has won, start new game --
                                                case 1: done = 1;
                                                                break;
//--computer player has made its move,
//now awaiting the user's turn
                                                case 0: break;
                                        }
                        }
                }
        }

        return 0;

}
```

The SDL library files, that were written by Sam Lantinga, are included on a CD. The code of those other files than images.h, engine.h, OP_Maze_RP.cpp was NOT written or modified by me.

Appendix C

The file Readme.txt contains information and direction on what the contents of the CD is.

The folder OP_Maze_RP contains all the source code, the library files, the dll files, the Visual C++ studio project file, etc.

The folder Game contains only executable, graphics folder (folder with images) and dll files in order to play the actual game – a zipped folder like this was sent to users to test the game. The folder Game too contains a Playme.txt file that gives a more detailed description of how the game should be played.

The file sdlvisualc.pdf contains a manual on setting up SDL with Visual Studio C++ 2005 Express Edition.

There will be Interim Report, Presentation, and Final Report (this report) included in Materials folder.