|  |  |
| --- | --- |
|  | ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ  ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ  ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ |

Ανάπτυξη και σχεδίαση προγράμματος σκάκι

|  |  |
| --- | --- |
| Καθηγητής |  |
| Μαθητής | Κωνσταντίνος Μαλωνάς |

Contents

[Εισαγωγή 3](#_Toc6737395)

[Requirements 4](#_Toc6737396)

[UML Analysis 9](#_Toc6737397)

[Pieces Movement Analysis 10](#_Toc6737398)

[The King 10](#_Toc6737399)

[The Pawn 10](#_Toc6737400)

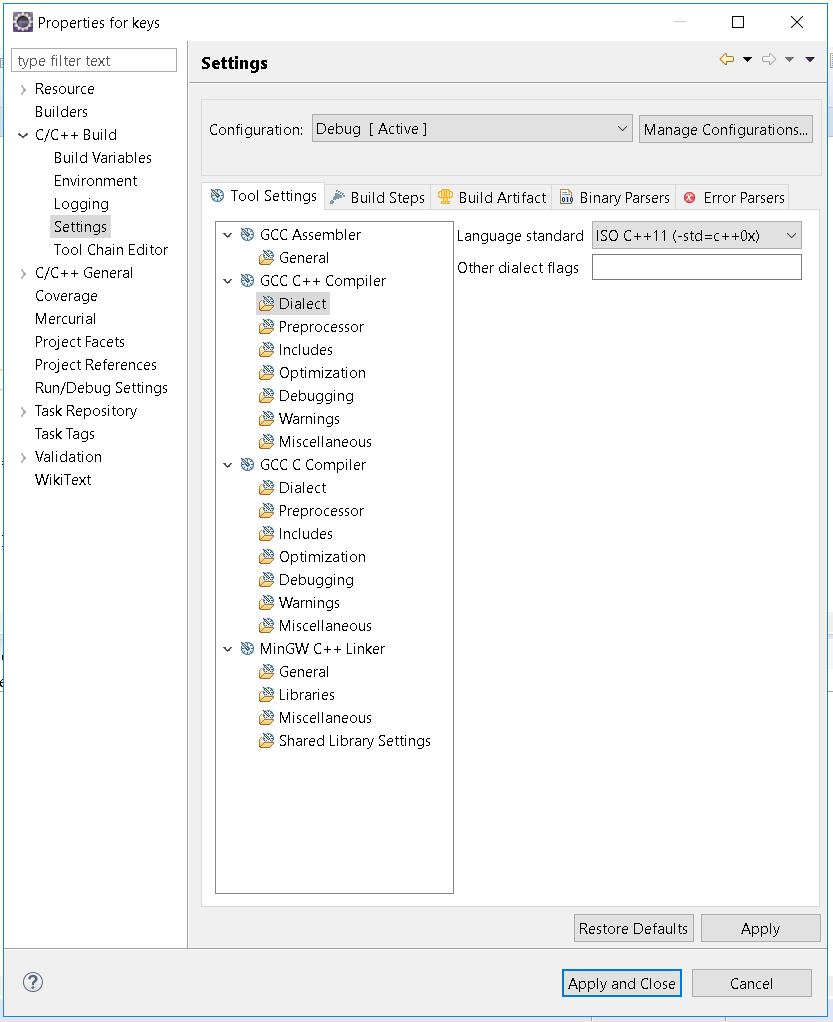
[References 11](#_Toc6737401)

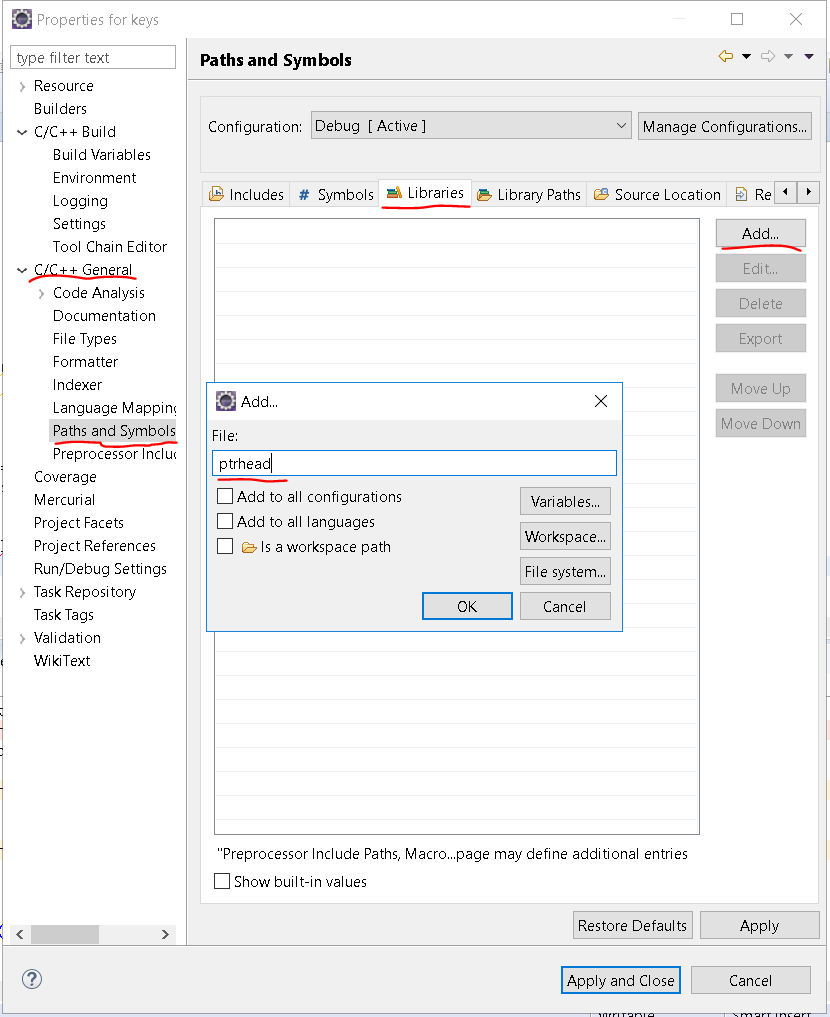
# Εισαγωγή

cin.get() is a synchronous call, which suspends the current thread of execution until it gets an input character (you press a key).

You need to run your loop in a separate thread and poll the atomic boolean, which you change in main thread after cin.get() returns.

It could look something like this:





https://www.quora.com/Why-could-the-thread-type-not-be-resolved-in-Eclipse-C++-even-when-the-compiler-doesnt-give-any-error-on-including-thread

# Requirements

The requirements were separated into functional and non-functional requirements [1]. Functional requirements refer to program characteristics that define what the application can do, e.g a functional requirement is that the program needs to be able to allow the player to move the pieces, and non-functional requirements refer to attributes of the program that are not related with what the program can do, but with how the program operates. For example, the choice of language that the application uses to print messages in the console, is a non-functional requirement. Choosing to output messages in Italian instead of in Greek, or English would not change the functionality and what the application does, in any way, but it would change its usability, as non-Italian speakers would find it difficult to understand how to use it. To be more precise, non-functional requirements (NFRs) can be defined as quality attributes (e.g., usability, reliability, security) or general system constraints [2].

Table 1, provides a list of the functional requirements extracted by the exercise specification:

|  |  |  |
| --- | --- | --- |
| Requirement  number | Functional Requirements | Covered |
| 1 | Έκτέλεση παιχνιδιού δύο παικτών | **Covered** |
| 2 | Το πρόγραμμα δέχεται από την standard είσοδο την κίνηση του κάθε παίκτη : | Το πρόγραμμα μας δέχεται input από το keyboard χρησιμοποιώντας το cin Με βάση το παραπάνω requirement η είσοδος διαβάζεται χρησιμοποιώντας το C++ object cin της istream class, το οποίο δέχεται είσοδο από το standard input device, στην περίπτωση μας το keyboard – [1] <https://www.programiz.com/cpp-programming/library-function/iostream/cin>. . |
| 3 | Το παιχνίδι ελέγχει την νομιμότητα της κάθε κίνησης και αν είναι νόμιμη την εκτελεί. | **Covered** |
| 4 | Σε κάθε κίνηση το πρόγραμμα ελέγχει αν υπάρχει νικητής ή ισοπαλία | **Covered** |
| 5 | Το πρόγραμμα καταγράφει τις κινήσεις των παικτών και αν ζητηθεί μπορεί να επαναλάβει τις κινήσεις της παρτίδας. | Not covered. |
| 6 | Αποθήκευση της τρέχουσας κατάστασης του ταμπλό σε αρχείο (savegameoption) | **Not covered** |
| 7 | Διάβασμα κατάστασης ταμπλό από αρχείο. Μετά το διάβασμα του αρχείου θα ακολουθεί εκτέλεση του παιχνιδιού (loadsavedgameandresumegameoption) | **Not covered** |
| 8 | Η κίνηση “En passant” δεν θα υλοποιηθεί. |  |
| 9 | Tο πρόγραμμα θα αναγνωρίζει “ισοπαλία” μόνο σε περίπτωση “stalemate”. |  |
| 10 | Η αποθήκευση τρέχουσας κατάσταση ταμπλό θα γίνεται μόνο πριν από κίνηση του λευκού. |  |

Πίνακας 1 - Functional Requirements

Table 2, provides a list of the non-functional requirements extracted by the exercise specification:

|  |  |  |
| --- | --- | --- |
| Requirement number | Non-functional requirements | Covered |
| 1 | Όλα τα μηνύματα του προγράμματος θα είναι γραμμένα στα Ελληνικά. | **Covered** |
| 2 | Για την παράσταση των κομματιών στο ταμπλό και στα αρχεία θα χρησιμοποιηθεί ο Αγγλικός συμβολισμός σύμφωνα με τα αναγραφόμενα [εδώ](https://en.wikipedia.org/wiki/Algebraic_notation_(chess))[3]. | **Partially covered** |
| 3 | Τα λευκά κομμάτια θα είναι γραμμένα με κεφαλαία και τα μαύρα κομμάτια με πεζά γράμματα. Εναλλακτικά, στην οθόνη μόνο, μπορούν να χρησιμοποιηθούν οι αντίστοιχοι unicode χαρακτήρες. | **Covered** |
| 4 | Μετά από ανάγνωση ταμπλό από αρχείο, το πρόγραμμα θα θεωρεί ότι είναι σειρά του παίκτη με τα λευκά κομμάτια να κάνει κίνηση. |  |
| 5 | Κάθε κλάση θα υλοποιηθεί σε ξεχωριστά αρχεία από τις υπόλοιπες |  |
| 6 | Το κυρίως πρόγραμμα θα υλοποιηθεί σε ξεχωριστό αρχειο – applicationfile. | Covered |
| 7 | Για κάθε κλάση θα δημιουργηθεί ένα header αρχείο το οποίο θα περιέχει την περιγραφή της | Covered |
| 8 | Για κάθε κλάση θα δημιουργηθεί ένα cpp αρχείο το οποίο θα περιέχει την υλοποίησή της | covered |
| 9 | Δεν επιτρέπεται η χρήση inline συναρτήσεων σε κανένα σημείο του υλοποίησή προγράμματος. | covered |
| 10 | Δεν θα χρησιμοποιηθούν ολικές (global) μεταβλητές. | Covered |
| 11 | Η αποθήκευση του ταμπλό θα γίνεται σε binary αρχείο. |  |
| 12 | Για κάθε κομμάτι που βρίσκεται στο ταμπλό θα γράφεται μία εγγραφή. |  |
| 13 | Η εγγραφή θα αποτελείται από τέσσερα πεδία του ενός byte:   * Την γραμμή του ταμπλό στην οποία βρίσκεται το κομμάτι * Την στήλη του ταμπλό στην οποία βρίσκεται το κομμάτι * Το γράμμα το οποίο συμβολίζει το κομμάτι (κεφαλαίο για λευκό κομμάτι, πεζό για μαύρο κομμάτι) * Την τιμή 1 αν το κομμάτι έχει κινηθεί ή την τιμή 0 αν το κομμάτι δεν έχει κινηθεί. |  |
| 14 | Η γραμμή και η στήλη είναι ακέραιοι αριθμοί με τιμές [1..8]. | Covered |
| 15 | Είναι επιθυμητό να γίνεται χειρισμός της κονσόλας ώστε, για παράδειγμα, ο χρήστης να επιλέγει κίνηση με τα “βελάκια” και το “Enter” και όχι πληκτρολογώντας τις συντεταγμένες του κομματιού που θα κινηθεί. |  |

Πίνακας - Non Functional Requirements

* Έκτέλεση παιχνιδιού δύο παικτών. Το πρόγραμμα δέχεται από την standard είσοδο την κίνηση του κάθε παίκτη, ελέγχει την νομιμότητά της και αν είναι νόμιμη την εκτελεί. Στην συνέχεια διαβάζει την επόμενη κίνηση, κ.ο.κ. Σε κάθε κίνηση το πρόγραμμα ελέγχει αν υπάρχει νικητής ή ισοπαλία.

# UML Analysis

# Pieces Movement Analysis

## Movement Overview

Each move is handled in a three-layer manner.

### Level 1 – Create Coordinates

To begin with, the method createCoordinates(), requires from the user to provide the coordinates of the starting point and the coordinates of the destination point for the move to be made. After the user provides this information, which is stored into corresponding variables, a check is made within the createCoordinates() method, to identify if the x,y origin coordinates provided correspond to a Square object that is of the appropriate Color. If that is the case, then the method identifyPiece() is called.

### Level 2 – Identify the Piece

The method identifyPiece() is ultimately responsible for checking what is the value of the field Piece, of the origin Square object, namely it checks of what type is the piece the player has selected to move. Though, before this process takes place, it is ensured that the coordinates provided by the player in the previous method, createCoordinates(), are actually within the bounds of the 8x8 chessboard, and that the starting and destination point do not contain Square objects (pieces) that are either both of Color BLACK, or both of Color WHITE. Then, by checking the value of the field Piece, of the Square object, the type of the piece is identified, and the corresponding method is called. For example if the Square object has a Piece that is KING,, then the moveKing() method is called, and the same happens for all the rest of the Square objects with other Piece values – except from when the Piece value is NONE, in which case the identifyPiece() method returns false and a relevant message.

### Level 3 – Move the Piece

### Rook

A chess piece, e.g. a rook, can execute as many steps as it wants, at any direction each time, if these steps are all horizontally or vertically and within the chessboard bounds. But it cannot move diagonally – that would be a bishop – and also it cannot move from position x to position x’, if between x and x’ there is another piece – a rook cannot jump above pieces. To get the behaviour required by a rook a number of checks are implemented within the moveRook() method. Firstly, it is checked  the case where the rook moves horizontally. The conditions for a horizontal advance are two. Specifically, the row should remain the same (row origin == row destination), and the column should not (column origin != column destination). For a vertical advance the two conditions are that the column should remain the same (column origin == column destination) and that the row should not (row origin != row destination). If none of these two cases abide with the move that the rook has been instructed to make the moveRook() method will return false, preventing the rook piece from executing a move that would be not rook-like.

## The King

The king can move in any direction.

## The Pawn

The pawn can move in four ways:

1. diagonally up one square if there is a piece of different color there
2. diagonally down one square if there is a piece of different color there
3. Forward one square if the square in front of it is empty
4. Forward two squares if it is a white pawn at column 1 or a black pawn at column 6 and the two squares in front of it are empty.

# References

[1] [https://stackoverflow.com/questions/16475979/what-is-the-difference-between-functional-and-non-functional-requirement](https://stackoverflow.com/questions/16475979/what-is-the-difference-between-%20%20%20%20functional-and-non-functional-requirement)

[2]<https://www.sciencedirect.com/topics/computer-science/nonfunctional-requirement>

[3] <https://en.wikipedia.org/wiki/Algebraic_notation_(chess)>