Bilkent University

CS319 Term Project

Design Report
Section 3

Team Name: JOKERS Team Project Name: IQ Puzzler Pro Project Group Members:

- Burak Erdem Varol
- Ravan Aliyev
- Subhan Ibrahimli
- Ismayil Mammadov
- Mahmud Sami Aydin
- Cihan Erkan

1.Introduction **3**
1.1 Purpose of the system 3
1.2 Design Goals 3
1.2.1 Trade-Offs 4
1.2.2 Criteria 4
2. High-level software architecture **5**
2.1 Hardware/Software Mapping 5
2.2 Persistent Data Management 6
2.3 Access Control & Security 6
3. Low-level design **7**
3.1 Final Object Design 7
3.1.1 Node Class 8
3.1.2 Polymino 8
3.1.3 PolyminoList 9
3.1.4 Level 9
3.1.5 GameEngine 10
3.1.6 Player 10
3.2 Packages 11
3.2.1 New Packages 11
3.2.1.1 Pieces Package 11
3.2.1.2 Game Package 11
3.2.1.3 LevelUtilities Package 11
3.2.1.4 UserInterface Package 11
3.2.2 Existing Packages 11
3.2.2.1 java.util Package 11
3.2.2.2 javafx.event 12
3.2.2.3 javafx.application 12
3.2.2.4 javafx.scene 12
3.2.2.4 javafx.scene.shape 12
3.2.2.5 javafx.scene.input 12
3.2.2.6 javafx.geometry 12
4. References **12**

1.Introduction

1.1 Purpose of the system

IQ Puzzler is a puzzle game which can be played as 2D and 3D. Purpose of our program by creating the digital version of the already existing physical game is to make the game more attractive by adding different modes and creating new level to satisfy the user. In addition to original version of the game there are two player,3D, and time modes. This game is planned to be multi-platform software ,which can be played on Microsoft Windows,Mac OS and Linux. It will be user-friendly, new players will be able to learn how to play the game easily. This game will challenge the players and entertain them by improving their inductive reasoning aptitude.The player is expected to put pieces on the board in a logical way, in order to arrive at the correct solution of the puzzle.

1.2 Design Goals

Design is one of the important processes for creating the software. It helps to define to software solution to sets of problems [1]. These problems are mentioned as non functional requirements in our analysis report which will be clarified in our design report.describe. Designing goals for all aspects of the software for building is the main focus of this paper. Important design goals for our software is given in the following list:

1.2.1 Trade-Offs

Development Time vs Performance

We discussed about which program language are appropriate for our project in terms of development time and performance. Because Java is appropriate language for object-oriented programing and because it is hard to control heap in C++, we chose Java for development of this game.

Space vs Speed

In order to make our game faster, we decided to make a lot of different and related classes as much as possible to improve speed of game. Because this project has a lot of classes this will take a bit much space in memory.

Understandability vs Functionality

In our game, we used accustomed control keys for the sake of understandability. All users from different age levels can play this game without any tutorial for beginning

1.2.2 Criteria

Usability:

One of the prominent goals of this game is being usable for all ages. Because this game is designed for 4+ ages, it is easily understandable with accustomed control keys.

Apart from that, virtual one of this game is a bit dangerous for little children. They can swallow pieces of puzzle and it can have danger of death. However, because our game does not contain these pieces it does not contain danger.

Capacity:

This game will not require high PC features. We designed this game for all social classes. Hence, poor and rich classes can get this game from Internet.

Performance:

Game performance another prominent goal. Game will be useless if it has low game performance. In this case we have paid attention performance of this game. Hence, users can play the game without getting angry.

Portability:

As it is stated in capacity part, because this game does not require high PC features, it will be available for different devices.

Extendibility:

Create level part of this game is designed for extend features of this game according to user's claims. User can make difficult/easy levels with different colors of pieces.

2. High-level software architecture

2.1 Hardware/Software Mapping

From the point of software mapping, the implementation of the game will be in Java programming language. The usage software requirements for the game in personal computer is the existence of Java Runtime Environment which makes the application executable.

From the point of hardware mapping, IQ Puzzler will require a keyboard and a mouse in order to play the game. The user does not need to have a high performance computer to play

2.2 Persistent Data Management

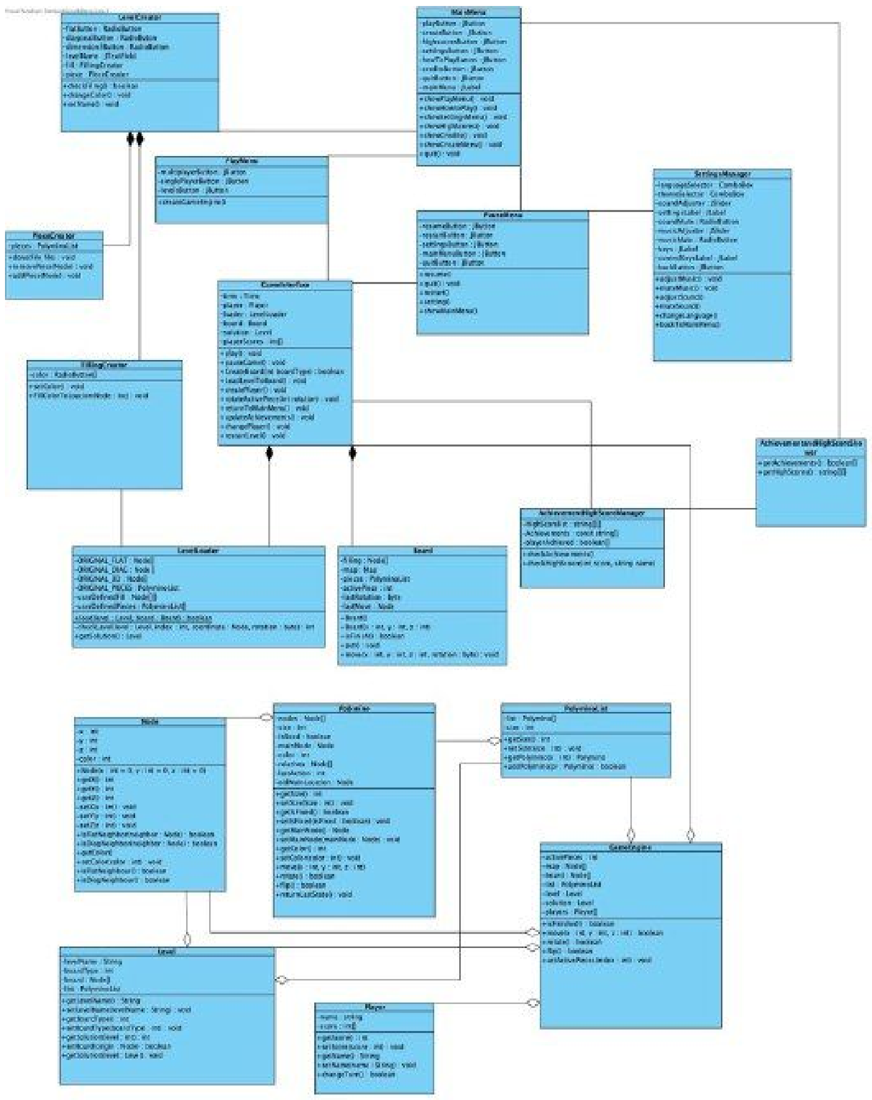
IQ Puzzler does not require many data storage. Game instances need to be stored in hard drive in order to demonstrate maps, objects, texts, or to keep the data for levels, high scores or to make functionalities of sound and so on. Some of these data are unchangeable to keep a visual of the game steady, but the others can be modified since it is dependent on the user interaction such as changing the language of the game, updating the current level or high scores, and keeping achievements

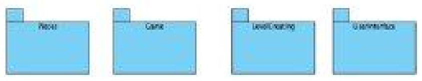
2.3 Access Control & Security

The user does not need to have an internet access in order to play game but there should be the executable application in order to play the game. Since it does not require any internet access and the user can play in local system, there is no risk obtaining any malicious actions.

3. Low-level design

3.1 Final Object Design





3.1.1 Node Class

Attributes :
int x : represents x coordinate of the node

int y : represents y coordinate of the node

int z : represents z coordinate of the node

int color: represents the color of the node

Constructor:
Node (int x, int y, int z, int color)

Methods:
boolean isFlatNeigbor(Node) : checks flat neighborhood with given node

boolean isDiagNeigbor(Node) : checks diagonal neighborhood with given node

3.1.2 Polymino

Attributes:
int size : number of the nodes in the polyomino

boolean isFixed : checks whether polyomino has fixed point on the board

Node mainNode: indicates the coordinate of the polyomino

int color : represents the color of the polyomino

Node[ ] relatives : represents the relative nodes of the main node of the polyomino

int lastAction : shows the last action was flip, rotate or move

Node oldMainLocation : holds position of the main node before last move

Constructor:
Polymino(int size, Node mainNode, Node[] relatives)

Methods:
boolean move(int x, int y, int z) :moves the polyomino to the coordinate according to x,y,z and returns boolean to show whether successful

boolean rotate( ): if polymino is rotated on XY plane (90 degree) , return true

boolean flip(): flips polyomino on Z axis (90 degree) , and return true if successful

3.1.3 PolyminoList

Attributes:
Polymino[] list : contains polyominoes for the level

int size : number of the polyominoes

Constructors:
PolyminoList(int size)

PolyminoList(int size, Polymino[] list)

Methods:
Polymino getPolymino(int x) : return the polymino in the list with index x

Boolean addPolymino (Polymino pl) : adds polymino to the list and returns true if it is done successfully

3.1.4 Level

Attributes:

string levelName: stores the name of the level also it provides file name in the system

int boardType: indicates the board type: flat , diagonal or 3d

Node[] board: indicates the coordinate of the board on the game engine

PolyminoList list: Polyominoes which does not contain (-1,-1,-1) coordinate, are the fixed ones at the beginning of the level. The ones with (-1,-1,-1) coordinate will be shown randomly on the game map.

Construtor:
Level(File level)

Level(PolyminoList list, string name, int boardType)

Methods:
int getSolution(int level): finds number of solutions for the level recursively and returns it

Level getSolution(Level level): returns a solution for the level recursively

boolean setBoard(Node origin): sets the coordinates of the board and returns true if successful

3.1.5 GameEngine

Attributes
int activePieces: indicate the index of which polyomino to move, flip or rotate

Node[ ] map: Coordinate system for the game

Node [ ] board: Board which should be filled by pieces

PolyminoList list: contains list of the pieces

Level level: indicates which level to play

Level solution: solution of the level for the 2 player game

Construtor:

GameEngine(Level level)

Methods:

boolean isFinished( ): returns the boolean to show whether the game is finished

boolean move(int x,int y, int z): moves the polymino according to given parameters and returns whether is successful

boolean rotate( ): rotates the polymino and returns whether is successful

boolean flip( ): flips the polymino and returns whether is successful

3.1.6 Player

Attributes
int score: indicate the score of the player

String name: indicates name of the player

boolean isTurn: indicates if it is player’s turn.

Construtor:
Player(string name)

Methods:
boolean changeTurn( ): makes it other players turn.

3.2 Packages

For our implementation, we are going to define our packages for main components of our game. However, for visual aspects of our game, we are going to use JavaFX libraries.

3.2.1 New Packages

3.2.1.1 Pieces Package

This package contains basic pieces, that user moves and places on board, of IQ Puzzler Pro and determines basic behaviour of them.

3.2.1.2 Game Package

This package contains the main parts that makes game playable. Interaction of pieces with each other and game environment will be defined in this package.

3.2.1.3 LevelUtilities Package

This package includes classes related to levels, such as creating, storing and loading.

3.2.1.4 UserInterface Package

This package contains everything related to GUI of the game.

3.2.2 Existing Packages

3.2.2.1 java.util Package

We are going to use this package to be able to use main functionalities of Java such arrayLists and Timer objects.

3.2.2.2 javafx.event

We are going to use this package for event handling including key and mouse events

3.2.2.3 javafx.application

We are going to use this package to initialize our application.

3.2.2.4 javafx.scene

We are going to use this package to present our game environment. It provides specifically cameras, groups, shapes, input and transformations.

3.2.2.4 javafx.scene.shape

We are going to use 3D shapes such as box and spheres to represent board and the pieces.

3.2.2.5 javafx.scene.input

This package provides key and mouse listener to interact with player

3.2.2.6 javafx.geometry

This package provides bounds of object in the application.[2]

4. References

[1] “Software Design.” *Wikipedia*, Wikimedia Foundation, 5 Nov. 2018, en.wikipedia.org/wiki/Software_design.
[2]“Oracle Documentation.”, 9 Dec. 2013, docs.oracle.com/javafx/2/api/overview-summary.html.