CS 319 - Object Oriented Software Engineering

Design Report

Iteration 2

Road Block

Group 1A

Denizhan Soydaş
Deniz Ufuk Düzgün
Kaan Atakan Öztürk
Cavid Gayıblı
Selimcan Gülsever
Serkan Delil

Contents Table

1. Introduction

1.1. Purpose of the System

Road Block is a game that we are planning to develop. The main objectives and game play of it are as follows: There is a thief in the game that is trying to escape from the police. The user's main objective is to block the paths which the thief may use in order to get away. The game is played on a map that offers 36(6x6) squares. This map offers various spaces in which the user can place differently shaped blocks and the puzzle in it is to place all the blocks where they have to be. These blocks are offered to the user at a seperate place from the game board and they can be chosen and placed in a random order as long as they are in the places where they have to be in the end. User is also able to, and usually should, rotate these blocks in order to find the solution for that level. The game consists of different levels which increase in difficulty after each one is completed. There are 3 difficulty levels in the game: Easy, Medium, Hard. Each of these difficulty levels have 10 levels inside meaning, our game consists of 30 levels in total.

1.2. Design Goals

We are required to decompose our system into pieces but before doing that, it is an important aspect for our project to briefly give information about our design goals. These design goals are mostly inherited from the non-functional requirements that we shared in the analysis report that we submitted before. First of our non-functional requirements was “Game Performance”. We wanted to implement a game that does not require relatively high specs and shared certain system specs about our performance goals. Those specs being: Ram 128 MB or higher; Disk Space 56MB; Graphics Card : 64 MB Ram or higher. We have achieved to run our game with the given specs. The second non-functional requirement was our game to be “User Friendly “. To test this requirements, we have had 40 individuals from various ages. To be more precise here are the age range of the user and the average score they gave for our game’s User Friendliness:

7-11 years old: 9 test users (Score average : 9.5)

12-22 years old : 12 test Users (Score average : 9.9)

23-40 years old : 10 test Users (Score average : 9.8)

41-75 years old : 9 test Users (Score average : 9.2)

Finally, our last requirement was Response Time and we have set a goal for our game to have around 40-50 ms. Our measurements during the gameplay showed that on average, we have about **14 ms** response time. Thus, it is safe to say that the smooth gameplay we aimed to have is obtained.

1.3. End User Criteria:

1.3.1. Simulation of real world

We are implementing a game. Therefore, our main goal is providing entertainment to the user. As one might guess, if we are to offer entertainment, our players should not have difficulties while grasping the basics of game the nor while playing it. With this idea, we have provided user friendly interfaces menus on which the user is able to easily reach wherever they want and perform the desired operations throughout all of the available interfaces. During the gameplay we have decided that our input device to be the mouse. User will be able to easily click and drag the blocks to the necessary places after choosing the right rotation for the blocks, again by using the mouse. This way, we will obtain simplicity for user's benefit.

1.3.2. Basic and applicable:

As in all of the games out there, a player should know how to play the game first. Therefore, we will provide an instructions manual involving how to rotate the blocks, how to move them, what is necessary for a level to be completed. This manual will also contain useful information about the objects of our game.(the thief, police car blocks, map etc.)

1.4. Maintenance Criteria

1.4.1. Modifiability

In our system it needs to be easy to modify the existing functionalities of the system to let it accommodate new ones. More specifically, having completed the first iteration, to realize the changes brought by the second iteration, we aim to make the code “modifiable”. By mentioning “modifiability”, rather than any future development concerns, we are concerned with an implementation which minimizes changes to the existing functions when a new function/change is to be handled. That’s why we are aiming for “modifiable” (independent) code - in case anything goes wrong and this requires changes modifications to existing functions. This can be thought to be more of a precaution to prevent us from rewriting the existing code again in case we fail to integrate new capabilities.

1.5. Performance Criteria

Response Time : For games in general, it is so important that user’s requests and actions have responses as quickly as possible. Otherwise, users might easily lose interest for the game and get distracted. Therefore, we have planned a system that almost immediately responds to the player and also is offering enthusiastic sounds, visual components for entertainment purposes.

1.6. Trade-offs:

1.6.1. Modifiability vs Response Time

Modifiability requires extra calls in order to preserve an environment such that the project can be changed in the future(by adding new features) without decomposing the structure all over again. However, a quick response time requires low call count in order to

instantly respond to user input. Therefore, we will focus on a balance between these two features to both have the user satisfied and make our job easier in the future developments to our game.

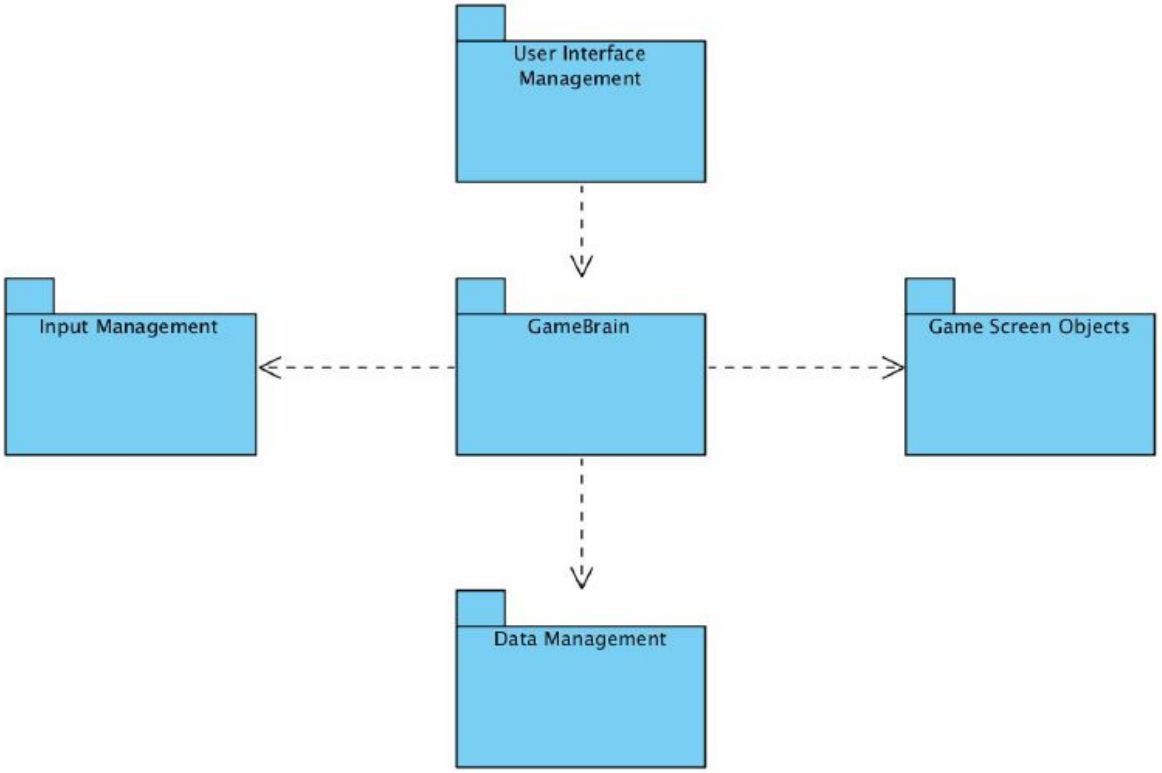
1.6.2. Functionality vs Learning Process (Ease of Use)

For games in general, it is an important feature that the user is able to easily grasp the basics of a game and how to play it. However, it is also expected by most of the customers that games have plenty of features and functionalities in order to “have fun” while playing the game. To conclude, even though these two topics are in conflict with each other, we are trying to offer a game environment that balances these two necessities of a “good” game.

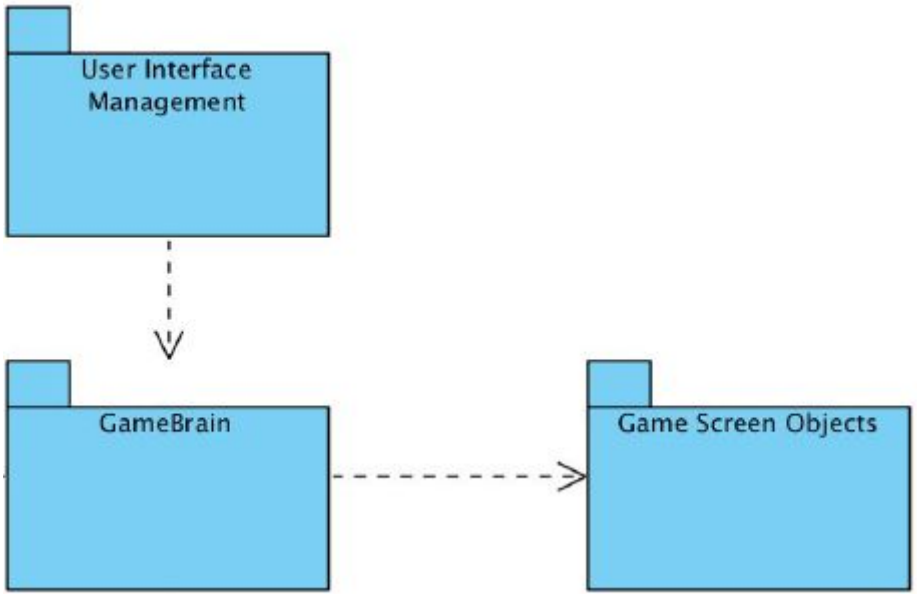
2. High-Level Software Architecture

2.1. Subsystem decomposition

Three-tier architecture is chosen to show structure of the project. Three-tier architecture consists presentation, process logic and data storage. Diagram that shows the project’s three-tier principle is given below as Figure 1. Presentation tier consists user interface and mainmenu, difficulty menu, levelmenu and all of their panels, buttons and components are at this tier. Depending on user’s choices on presentation tier,all signals pass from User Interface Management package to Gamebrain package.



To explain further actions, if the user clicks on the button “Play game” in main menu , program will go to our “DifficultyPanel” to let the user to choose a difficulty. After user’s choice in difficulty menu, levels are seen on the screen and user choice which level he/she want to play.Up to this part the process logic tier is used. Afterwards, with the help of “GameBrain” package and its helpers “Map” and “Entities” classes, the game will be set up with its logics. Then, the user will start playing the game. The data storage tier consists of classes that keeps the data needed for objects of the game.These datas are time for point calculation, status of collisions for checking whether game is finished or not, the coordinates of entities such as police, buildings .



2.2. Hardware/software mapping

Software: The game is implemented by using Java packages and libraries. The reason behind the selection of the Java language was its wide usage and compatibility with the all widespread OS variations. As long as Java Runtime Environment is installed, the game can be played on Windows, Mac and Linux platforms.

Hardware: To play the game, in terms of the hardware requirements, the player will need an ordinary computer with a mouse and screen. Furthermore, external audio output device like a speaker or headphones will be needed for the recommended gameplay and hearing the sounds.

2.3. Persistent data management

The game does not involve data storage for the level which user passes .It is because,there is no limitation for selecting levels in order to waste of time in basic levels. In

addition, images of the levels and entities will be stored in a small file that will be in the game's files.

2.4. Access control and security

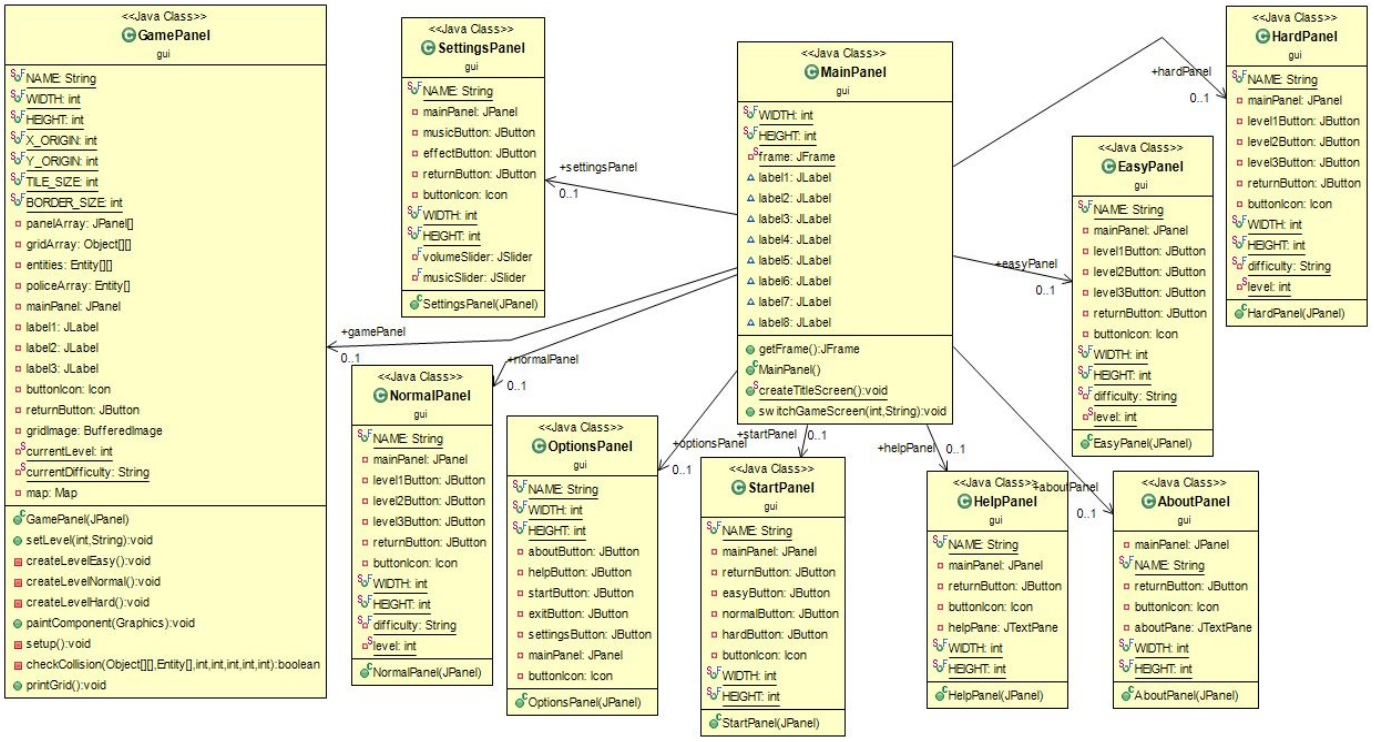
The game does not use any network-related authorization or registration system which will avoid the security issues that may happen because of it. Thats why, no measurement regarding access control and security needs to be taken

2.5. Boundary conditions

The game is to be launched using a ".jar" file. It will not require any prior installation. Also users need the JRE and JVM to execute the game. The game will be easily terminated by the help of "Exit" button on the main menu or X button of the window for the unsafe termination. Unsafe termination will not be affecting any fundamental game structure like source and player will be able to play again the level. Moreover, all menus except main menu have "returntomenu" button. Therefore, user can easily return to menu by clicking that button. When the game finished, user can try again or return to menu. If the game crashes somehow, user can play the same level where he/she trying to finish.

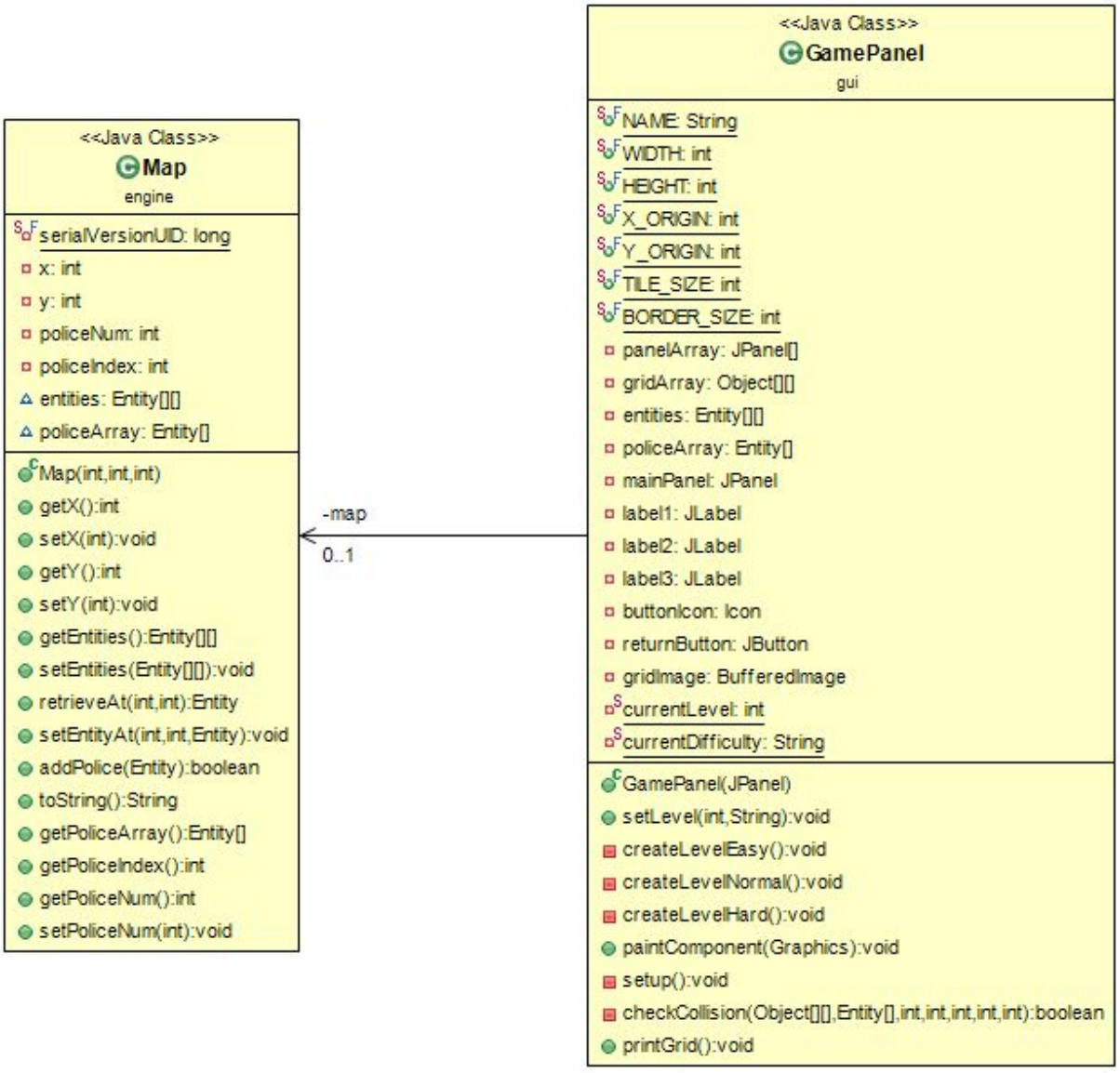
3. Subsystem Services

3.1. User Interface Subsystem



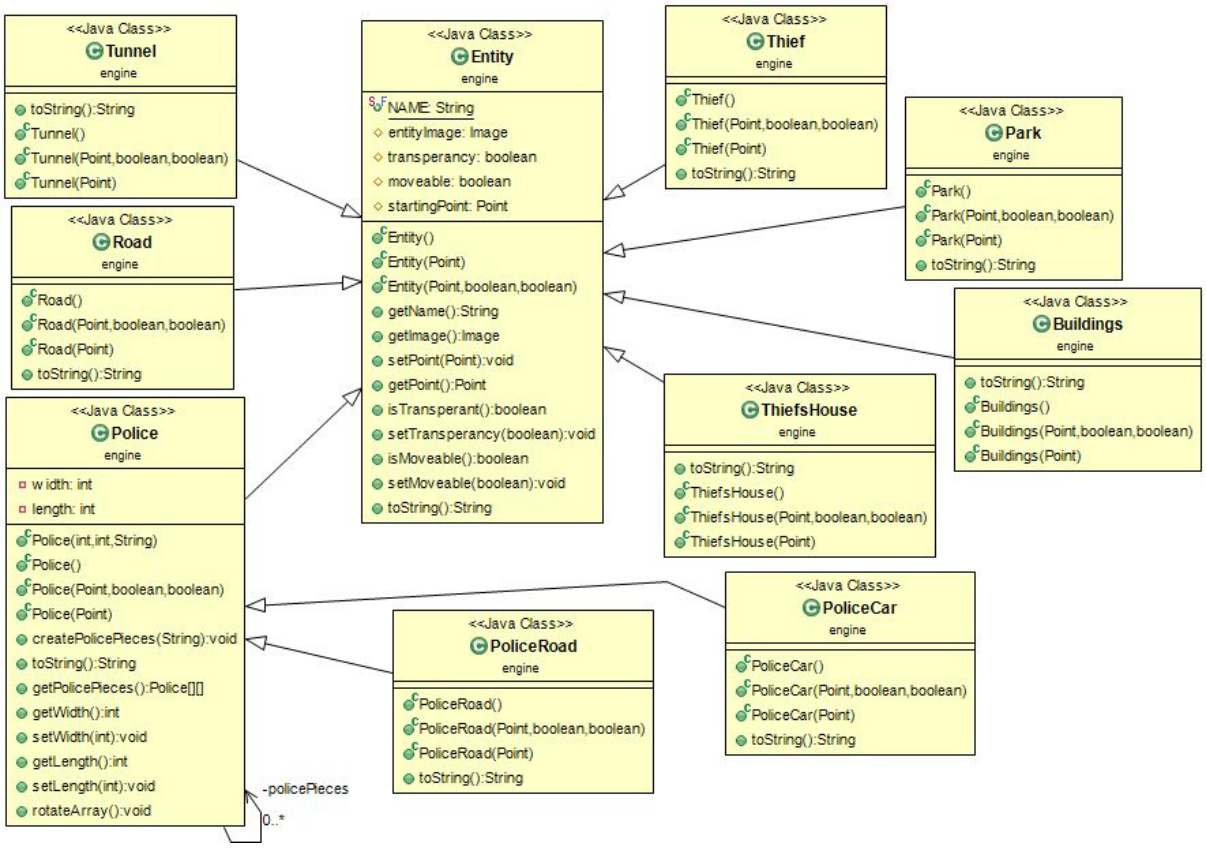
The User Interface (UI) will be in charge of the visual game elements. It will be dealing with the view of the game.This subsystem includes classes that are MainPanel, SettingsPanel, StartPanel, NormalPanel,EasyPanel,HardPanel,AboutUsPanel, HelpPanel, OptionsPanel, GamePanel. MainPanel contains all the panels. Furthermore, GamePanel is part of user interface subsystem and game brain subsystem since it includes design and logical methods.

3.2. Game Brain Subsystem



Game Brain Subsystem is responsible for creating and updating maps of the game. This subsystem also checks whether game is finished. In addition, all calculations about score of user will be done in Game Brain Subsystem. Furthermore, Game Brain Subsystem contains GamePanel and Map classes.

3.3. Game Screen Objects Subsystem



Game Screen Objects Subsystem decides objects that are shown in the screen while the game starts to running. GamePanel will draw entities in this subsystem. These entities are Buildings, ThiefsHouse, Tunnel, Thief,Park and Police. This subsystem has also important methods about police. For example, rotate function is in the Police class.

3.4. Data Management Subsystem

In the project, we are going to use serialization. We will read serialized maps from the hard disk to load the architecture of each level. Instead of initializing whole the map entity array by putting each element on it, we decided to keep these maps as serialized objects in hard disk. For more information, please look at:

https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html

4. Low-level Design

4.1. Our group's solutions to object design trade-offs

As our trade-offs were discussed in our "Design Goals" section, we would like to mention solutions that we came up with for dealing with these trade-offs in "low-level" terms.

4.1.1. Response Time vs Modifiability

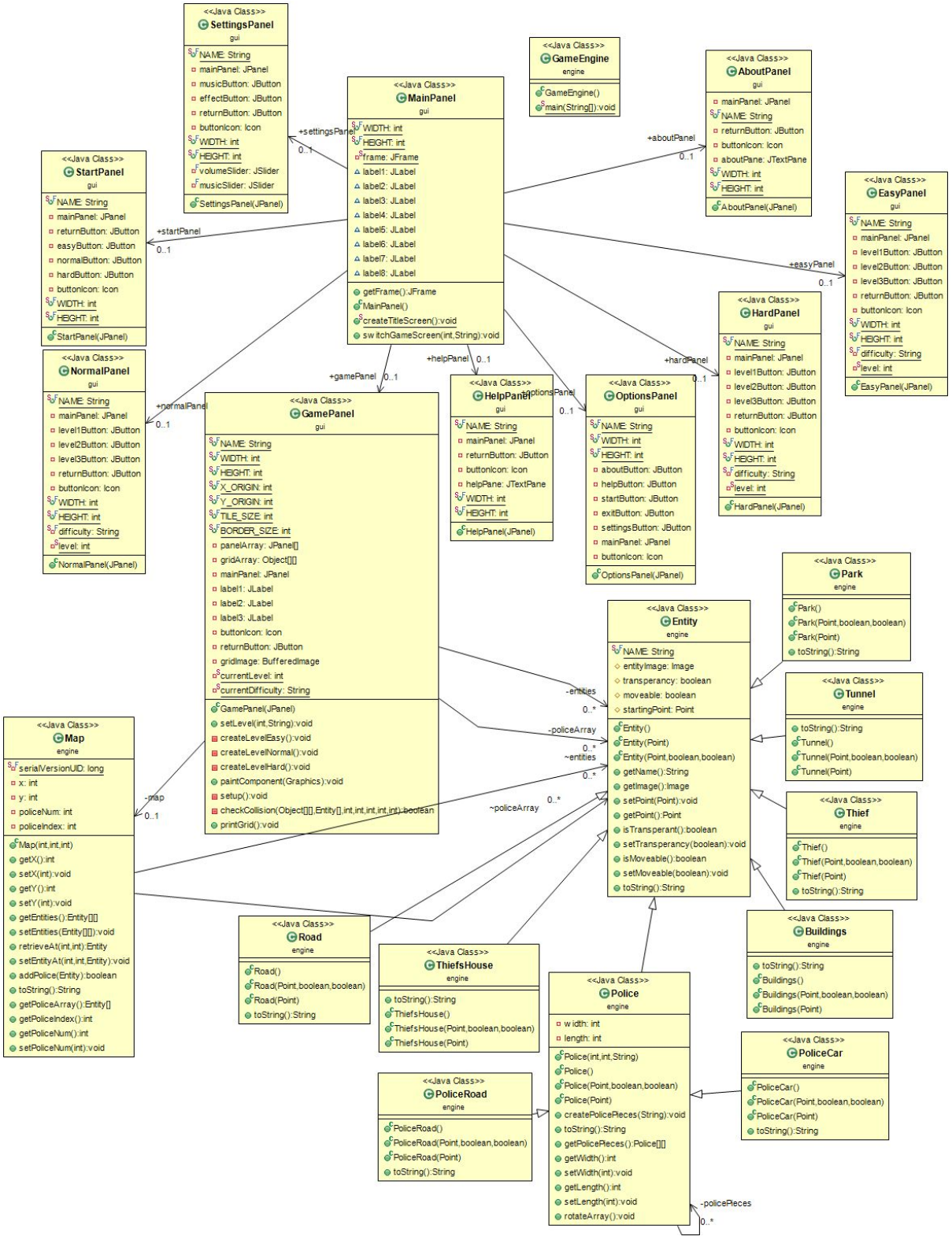
We are implementing a game.Therefore, response time is one of the most important concerns with our project because we need to respond as quickly as possible to maintain a "fun" environment and prevent possible causes that might lead to loss of interest of the users. One of the ways to do so is, in our object design, providing direct references to objects, especially those who need a faster access time. By this way, we tried to minimize the number of unnecessary calls. These unnecessary calls would cause a slower overall response time especially for the core classes of our game. To also handle with modifiability, we have tried to make our functions as independent as possible from each other. There are of course functions that depend on each other but we have tried to minimize the amount of these functions to improve modifiability for possible changes, adjustments in the future.

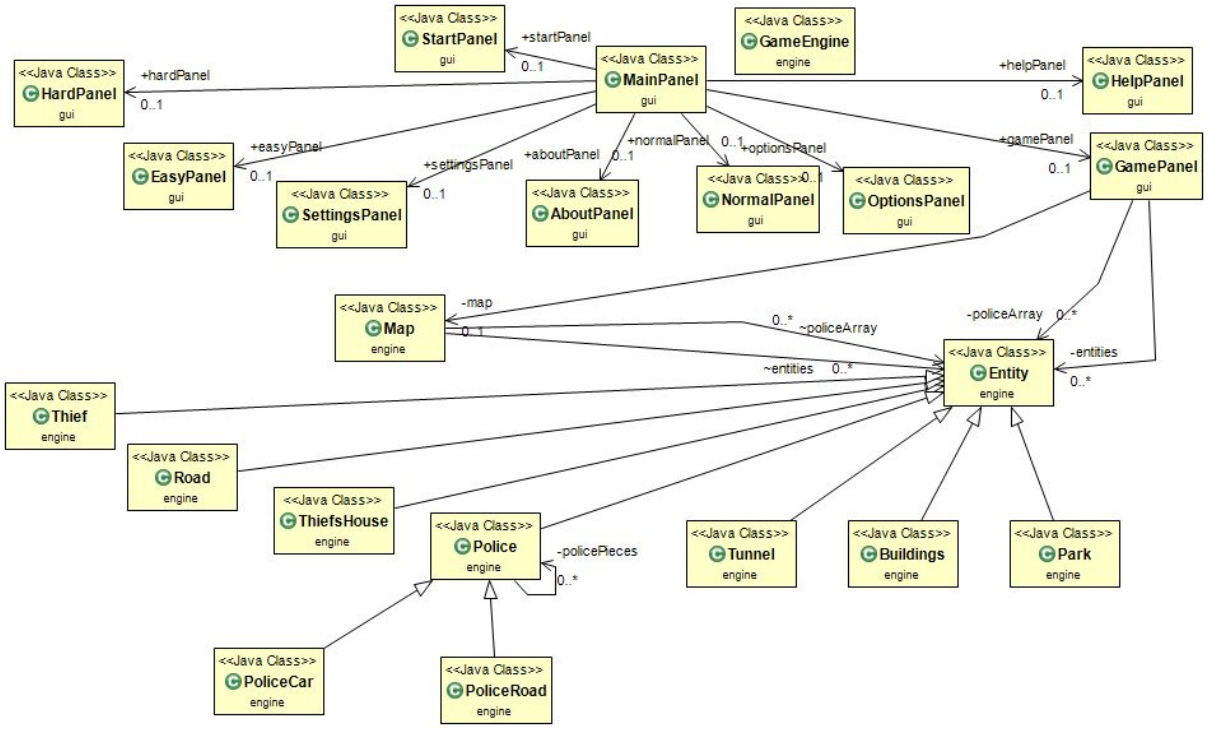
4.1.2. Memory vs Response Time

We are aware that memory was not specified in "Design Goals" part of our report. However, consuming more memory means a slowdown for the response time since it conflicts with speed. Since one of this course's main goals is to learn about and practice with the interactions between the classes, we have not really cared about the memory

consumption. Instead of prioritizing memory, we focused on creating as many classes as possible to practice with the interactions in between them. It is in our attention that our project is relatively a small one so our system will not consume that much of a space in a memory drive. However, we still felt the need to share about the memory planning that we had. In our project, in order to obtain required objects we possess a variety of children classes that can be refactored out to a larger, enumerated type-using, parent class(single).

4.2.Final object design





Diagrams above, include our projects final object designs,with all of our classes while specifying their names and methods.

Game Panel

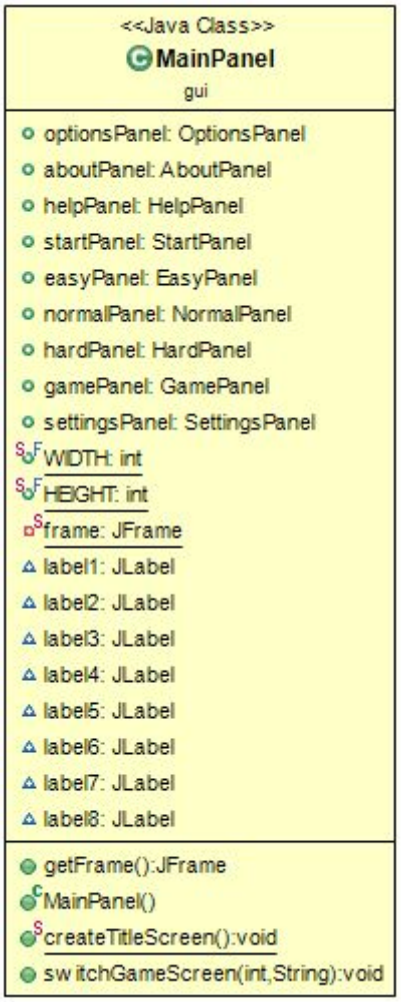
Attributes

- **NAME: String :** Name of the objects such as police and thief.
- **WIDTH: int :** Width of the different objects like police, thief, building and etc.
- **HEIGHT: int :** Height of the different objects like police, thief, building and etc.

- **X_ORIGIN: int :** X is representing the location wherever particular objects exists.
- **Y_ORIGIN: int :** Y is representing the location wherever particular objects exists.

Functions

- **setup(): void :** This function sets up all the map with given attributes.
- **checkCollision(Object[][], Entitiy[], int, int, int, int, int) boolean :** This function checks whether there are some objects on top of other.



Main Panel

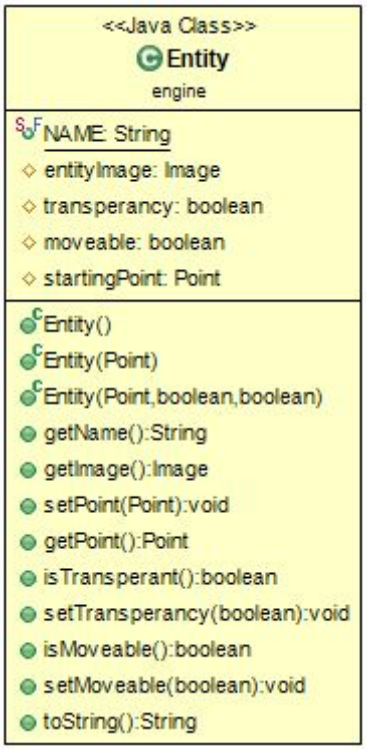
Attributes

- **width :** The width of the Main panel screen.
- **height:** The height of the Main panel screen.
- **frame:JFrame :** Frame of the Main Panel screen.
- **label1:JLabel :** first label of the Main Panel
- **label2:JLabel :** second label of the Main Panel
- **label3:JLabel :** third label of the Main Panel
- **label4:JLabel :** fourth label of the Main Panel

- **label5:JLabel :** fifth label of the Main Panel
- **label6:JLabel :** sixth label of the Main Panel
- **label7:JLabel :** seventh label of the Main Panel
- **label8:JLabel :** 8th label of the Main Panel

Functions

- **getFrame() :** This function gets the frame of the JPanel..
- **MainPanel() :** This function creates the main panel.
- **createTitleScreen() :** This function creates title screen of the Main Panel.
- **switchGameScreen() :** This function switches the current screen of the game.



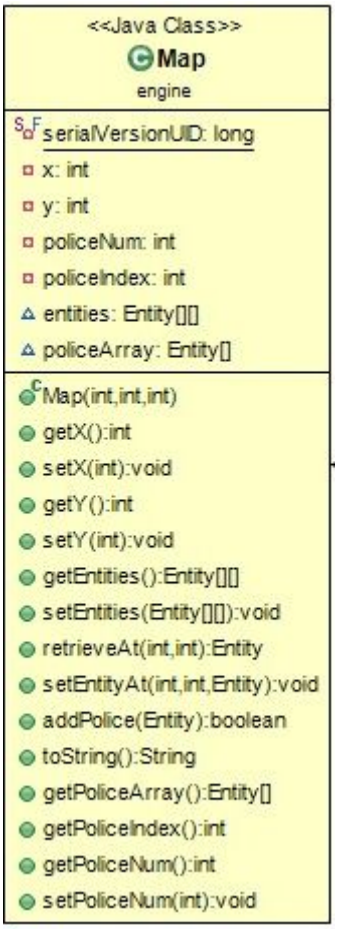
Entity

Attributes

- **transperancy: boolean :** This attribute indicates that whether the objects capable of overcome the obstacles in the game.
- **Moveable: boolean :** This attribute determines the status of the objects whether they are moveable or not.
- **startingPoint: Point:** This attribute determines the starting point of the objects.

Functions

- **Entity(Point, boolean, boolean) :** The constructor that takes parameters and will be used to generate entity objects.
- **getImage(): Image :** This function gets the Image of the objects.



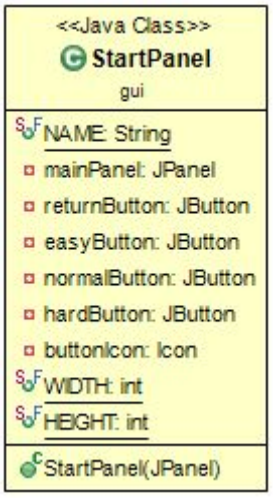
Map

Attributes

- **x: int :** x is the vertical coordinate of the map
- **y: int :** y is the horizontal coordinate of the map
- **policeNum:int :**policeNum is the number of the Police in the game.
- **policeIndex:int:** policeIndex is the index of the police in the corresponding array.
- **entities: Entity[][] :** entities is the two dimensional array which is vertical and horizontal lengths.
- **policeArray:Entity[][]:**police is the two dimensional array which holds the entity objects

Functions

- **Map(int,int,int) :** Map is the constructor of the Map class.
- **getX(): int :** This function gets the vertical coordinate of the map.
- **setX(x: int) : void** This function sets the vertical coordinate of the map.
- **getY() : int :** This function gets the horizontal coordinate of the map.
- **setY(y: int) : void** This function sets the horizontal coordinate of the map.
- **getEntities() :Entity[][]:** This function gets the entity objects from respective array.
- **setEntities() :Entity[][]:void** This function sets the entity objects to the map.
- **retrieveAt(int,int):Entity** This function retrieves the entity objects which is located at corresponding coordinates.
- **setEntityAt(int,int,Entity):void:**This function sets the objects to the specified coordinates.
- **addPolice(Entity):boolean:**This function adds the police to the map.
- **toString():String:**This function prints the values to the screen.
- **getPoliceArray():Entity[]:**This function gets the array of the respective police object.
- **getPoliceIndex():int:**This function gets the policeIndex.
- **getPoliceNum():int:**This function gets the policeNum.
- **setPoliceNum(int):void:**This functioon sets the policeNum.



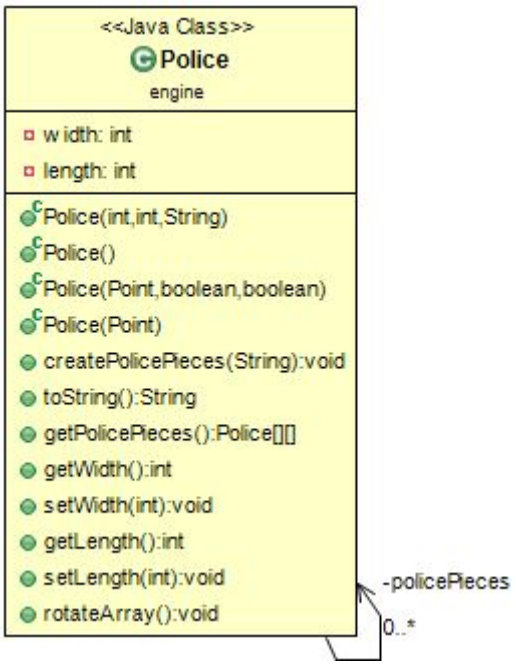
Start Panel

Attributes

- **name:String :** name of the panel.
- **mainPanel:JPanel:** mainPanel is the panel of the startPanel screen.
- **returnButton:JButton:**returnButton is the button to return to the menu.
- **easyButton:JButton:**easyButton is the button that enables to selection of the easy mode of the game.
- **normalButton:JButton:**normalButton is the button that enables to selection of the normal difficulty of the game.
- **hardButton:JButton:**hardButton is the button that enables to selection of the hard difficulty of the game.
- **buttonIcon:Icon:**buttonIcon is the image of the button.
- **Width:int:**width is the width of the panel.
- **Height:int:**height is the of the panel.

Functions

startPanel(JPane)():startPanel(JPanel) starts the game.



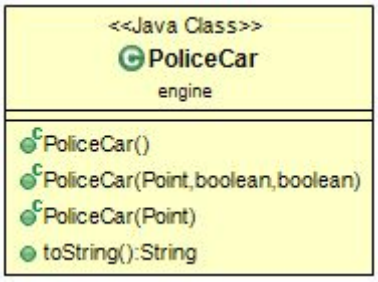
Police

Attributes

- **width: int :** The width of the police object (we use 1800 in this game).
- **length: int :** The length of the police object (depends on the shape of the objects).

Functions

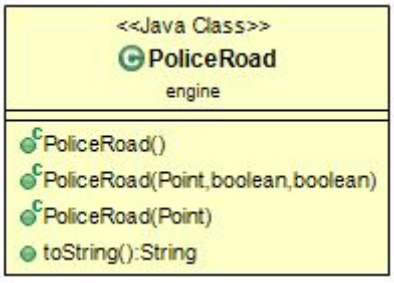
- **createPolicePieces(String): void :** This function creates police pieces in different shapes like T, L, or Z and etc.
- **rotateArray(): void :** This function rotates the police array by clicking the right click of the mouse.



Police Car

Functions

- **PoliceCar():** The constructor that will be used for generating objects
- **PoliceCar(Point, boolean, boolean):** The constructor that takes parameters and will be used to generate objects.
- **PoliceCar(Point)**: The constructor that will be used for generating objects.
- **toString():String:** toString method is the print function of this class.



Police Road

Functions

- **PoliceRoad():** The constructor that will be used for generating objects

- **PoliceRoad(Point,boolean,boolean):**The constructor that takes parameters and will be used to generate objects.
- **PoliceRoad(Point):**The constructor that will be used for generating objects.
- **toString():String:**toString method is the print function of this class.

4.3. Packages

4.3.1. Java Packages

4.3.1.1. javax.swing

Swing provides the look and feel of modern Java GUI. It is used to create graphical user interface with Java.

4.3.1.2. java.awt

AWT contains large number of classes and methods that allows you to create and manage graphical user interface (GUI) applications, such as buttons, windows, mouse listeners, scroll bars, etc. It is also used to change the color and font of the GUI.

4.3.1.3. java.io

Java IO is an API that comes with Java which is targeted at reading and writing data (input and output). For instance, read data from a file or over network, and write to a file or write a response back over the network.

4.3.1.4. java.imageio

ImageIO is a utility class which provides lots of utility method related to images processing in Java. Most common usage is reading from image file and writing images to file in Java.

4.3.2. Directories

Dividing the system into other subsystem provides a better and uncomplicated work, so MVC Directory Structure is used when our program was developed.

4.3.2.1. Model Directory

This directory contains all related classes for Entity class, and its subclasses.

4.3.2.2. View(GUI) Directory

This directory contains all related classes for the GUI of the program.

4.3.2.3. Image Directory

This directory contains all images used in the GUI such as background, shapes, etc. and the textures of the GUI.

4.3.2.4. Data Directory

This directory contains text files for different prepared levels in the game.

4.3.2.5. Controller Directory

This directory contains management classes for the game such as Game Engine, Sound Manager and Menu.