1. Personal information:

Reko Wenell
First-year student of Computer Science
Student ID: 785956
12.2.2020

2. General Description

The program is a turn based formula game where two players race against each other on a race track. Both players will control a car which has five different gears affecting both the maneuverability and the speed of the car. Each turn the players have a chance to adjust both gear and the direction of the car by one step before the move is issued.

As the gear is adjusted, the substituting direction is the one closest to the original direction.

The race starts with both cars on gear one facing left. As they drive, only one car may be in a single spot, which means a car trying to drive over another gets bumped and stopped. Similarly, a car can't drive over a wall and gets stopped if it tries. Finally, the car to cross the finish line first wins. In case of a tie, the player who has a bigger gear on wins. If there is still a tie, player2 wins as player1 has the advantage because he goes before player2.

As the program has the required functionalities and a graphic user interface it has been implemented in a moderate level of difficulty.

3. User interface

Once the program is started, a dialog will appear asking whether a player wants to choose an existing driver. If the user presses no, they will need to type a name no longer than 13 characters and press "OK". If they typed too long a name, a dialog will pop up to tell this, and the process will start from the beginning. The same thing happens if the user presses cancel. If the user chooses to use an existing driver, they will be directed to choose a .txt file from the /drivers folder of the program. After the first driver has been picked, the same process is repeated for a second one. Then a dialog asking the player to choose a .txt file will appear for the user to choose a race track from the program's /racetracks folder of the program.

Now a window with the game proper will appear before the user, filling the entire screen. The size of the window may be adjusted but not below a certain minimum size, where all the elements are still fully shown.

At the left side of the screen, the user will see some information about the car and the current situation. They will see what their currently chosen gear change and direction

change are, as well as whose turn it is and which round it is and what is the car in turn's current gear. In the middle of the page, they will see the game map with cars, road, offroad and finish line. Over the map there are printed the car in turn's possible destinations depending on his choices. These destinations do not, however, take into account any obstacles along way such as another car or offroad.

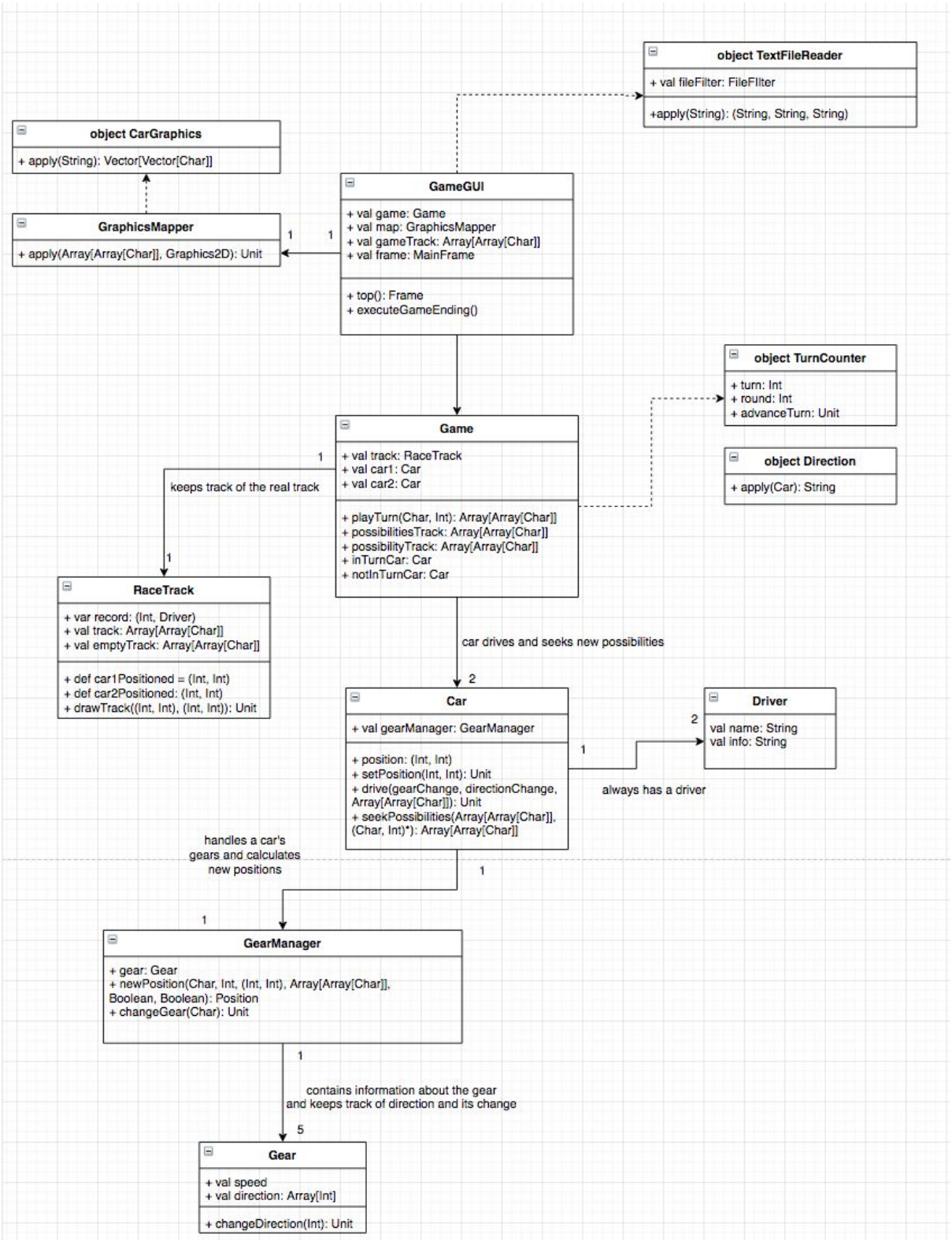
On the right the user sees three buttons. The uppermost, when pressed, pops up a dialog showing who holds the current record of this track and what the record is. The centermost confirms player's choices and drives the car forward, passing the turn to the next car. The undermost button makes the map show the same possible destinations as in the beginning: all nine if no overlapping and not taking into account any obstacles.

At the bottom of the window the user sees six buttons in two rows. The upper row is used for controlling the car's gear. If "Raise Gear" is pressed, the currently chosen gear change is changed into a '+' meaning the gear will be raised by one once user presses the "Confirm choices and drive"-button (the centermost of the three buttons on the right. "Keep Gear" similarly changes it to '=', which means keeping the same gear, and "Down Gear" changes it to '-', which means lowering the gear by one. After pressing one of these buttons, the map will also get updated: instead of all the possible destinations that don't even take obstacles into account, the user will see only the destination the car would reach if they now pressed "Confirm choices and drive". Any and all possible obstacles have been taken into consideration when calculating this destination, so it will be precise. As stated above, the user may press the "Show all option"-button in order to return to the view showing all possible destinations.

Players will then drive their cars one by one, until one of them crosses the finish line. The crossing of the finish line will only be checked after the second player's turn to give each player as many turns to reach the finish line. After that, the game is over and a dialog congratulating the winner will appear. The same dialog will also ask whether the user wants to save the information of these drivers. If yes, both drivers will either have their file altered to include this race (if they already have one) or get a completely new one (if they did not). These same drivers will then be available to choose in further races.

After that, the players may still look at the map, but the "Confirm choices and drive"-button will be switched off so that they cannot drive anymore.

4. Program structure


object TextFileReader
+ val fileFilter: FileFIlter
+apply(String): (String, String, String)
object CarGraphics
+ apply(String): Vector[Vector[Char]]
GameGUI
+ val game: Game
+ val map: GraphicsMapper
+ val gameTrack: Array[Array[Char]]
+ val frame: MainFrame
+ top(): Frame
+ executeGameEnding()
GraphicsMapper
+ apply(Array[Array[Char]], Graphics2D): Unit
1
1
object TurnCounter
+ turn: Int
+ round: Int
+ advanceTurn: Unit
Game
+ val track: RaceTrack
+ val car1: Car
+ val car2: Car
+ playTurn(Char, Int): Array[Array[Char]]
+ possibilitiesTrack: Array[Array[Char]]
+ possibilityTrack: Array[Array[Char]]
+ inTurnCar: Car
+ notInTurnCar: Car
object Direction
+ apply(Car): String
1
keeps track of the real track
1
RaceTrack
+ var record: (Int, Driver)
+ val track: Array[Array[Char]]
+ val emptyTrack: Array[Array[Char]]
+ def car1Positioned = (Int, Int)
+ def car2Positioned: (Int, Int)
+ drawTrack((Int, Int), (Int, Int)): Unit
car drives and seeks new possibilities
2
Car
+ val gearManager: GearManager
+ position: (Int, Int)
+ setPosition(Int, Int): Unit
+ drive(gearChange, directionChange, Array[Array[Char]]): Unit
+ seekPossibilities(Array[Array[Char]], (Char, Int)*): Array[Array[Char]]
1
2
Driver
val name: String
val info: String
always has a driver
handles a car's gears and calculates new positions
1
1
GearManager
+ gear: Gear
+ newPosition(Char, Int, (Int, Int), Array[Array[Char]], Boolean, Boolean): Position
+ changeGear(Char): Unit
1
contains information about the gear and keeps track of direction and its change
5
Gear
+ val speed
+ val direction: Array[Int]
+ changeDirection(Int): Unit

Class Game represents the internal game logic. It takes as parameter information about the track and the drivers. It first reads the information about the track, parses through it, and saves all significant details into variables. The Game then creates a RaceTrack, to whom it passes on the relevant information about the track, and two cars with drivers corresponding with the driver information. It also has several methods for calling information about the cars, a variable isOver, which ends a game if true and a variable including the winning driver.

The main methods of the Game class are threefold, but they all have similar purposes. The most important method is playTurn which represents playing a turn as in the car that is in turn driving. It is given a character ('-', '=' or '+' representing the player's desired changes to the gear and an integer (-1, 0 or 1) representing a player's desired changes to direction as parameters. It first calls the car that is in turn to drive according to playTurn's given parameters and the current RaceTrack. It then asks RaceTrack to update its track (drawTrack-method), giving the car positions to it as parameters. It then calls the next car to draw its own track with the seekPossibilities-method. This is the map that the method finally returns. Before that, however, it checks if anyone has won the game and advances the game by one turn if that is not the case.

Other important methods are possibilitiesTrack which calls the car in turn's seekPossibilities-method to get all possible destinations and possibilityTrack which calls the car in turn's seekPossibilities-method to get a single possible destination. The returnValues of these methods are quite similar to playTurn, but they do not alter anything in the programs state.

Class RaceTrack models the physical track. From the given string given as parameter at its creation, it creates an array of arrays that includes all the information about the track. It also reads the first positions of the cars when building the array and the Game object reads them from there. RaceTrack also has the drawTrack-method, which given the new positions of the cars, updates the array to correspond with them and also leaves markers behind to the positions the cars leave.

Class Car is given as parameters its driver, its first position (which it gets from RaceTrack through Game), its avatar (either A or B) and the finishing line (which it gets from RaceTrack through Game). It then creates a gearManager of its own. The three important jobs of a Car are A) keeping track of its position on the board. B) Using method seekPossibilities, which creates a track with the possible future destinations printed on the physical track. How this method precisely works is a bit different depending on how many parameters of (player's desired change to gear, player's desired change to direction) are given. In case of zero, it draws all the possibilities without taking any obstacles into consideration. In case of one or more, it draws the precise possibilities. This method does not affect anything in the program. It calls gearManager to check where the car would end up in. C) A Car can drive given player's desired change for gear, player's desired change for gear and the track. It calls GearManager to calculate the new position and then updates its own position.

Each Car has exactly one GearManager that takes care of its gear and direction, changes to them and calculates any future destinations. It takes as parameter the coordinates of the finish line. It has five gears and switches between them the currentGear-variable. It also keeps record of the laps the car has driven and the lap times. The important method of this class is newPosition, which does a lot, but basically tells where the car will drive with the given parameters. It is given as parameters player's desired change for gear, player's desired change for gear, the track, whether the car will actually drive or not and whether this is looking for a single future destination. When the method is called, GearManager changes gear and direction (through Gear) as per its parameters. According to its latter two parameters, it calculates where the car would end up in and returns the coordinates. If the car is not actually moving, it will revert everything back to how it was before.

Abstract class Gear has five different kinds of gears, each representing a gear of the car. These gears hold the logic of the car's direction and the actual value of the car's direction. They have one main method, which is directionChange and simply calculates and sets a new direction based on the player's desired change to direction.

Class Driver holds only the driver's name and information for writing a file about him/her later.

Object TurnCounter keeps track of the current turn and therefore also the round. Game and FormulaGUI objects need and use this information in determining which player's turn it is currently and sharing the information with the players. GearManager also needs the information about the rounds in order to calculate lap times. Game's playTurn method always updates TurnCounter's turn by one.

Object Direction's sole function is to take a car as a parameter and return a string that tells which of the eight directions the car is currently headed. It of course derives this information from the gear's direction information.

Object TextFileReader's apply-method takes a directory inside the project ("/racetracks" or "/drivers" as a parameter and opens a FileChooser Dialog where the user may choose a file. It then reads the file and returns a) its information without line breaks or spaces b) as read from the file and c) the file's pathname. Only FormulaGUI uses this object.

GraphicsMapper class with its apply-method sets up a given Graphics2D to look like a given array of array of characters (a track). These graphics will later be used to draw a picture that is the game map by FormulaGUI. The apply-method uses object Direction for the direction

the car faces and asks an array of array of characters representing pixels of the car from object CarGraphics to draw the cars in the right way.

FormulaGUI is represents the user interface of the program. It takes user input on the file representing the track the race is going to take place on and on the driver information passes it to a Game object when creating it. It takes user input on the player's desired gear change and the player's desired direction change and calls the Game's playTurn-method to with them as parameters to move the game forwards. When the game is over (which is finds out from the Game object's gameOver-method) it congratulates the winning player (finds out who exactly it is by the Game object's victoriousDriver-method) and saves information about the drivers as a file if the players wish it. This information includes lap times for drivers, so the method gets them from the GearManager of the Driver's Car that stores the information in a buffer. It also possibly updates the record holder of the track into the track's file.

5. Algorithms

When a Gear is changed, the GearManager must change the new Gear's direction into the closest one of the last gear. Now, the possible direction are a square whose side is equal to the gear's number. Here is an example of gear three's directions:

```
  -2, -1, 0, 1, 2
 2 X   X  X X X 2
 1 X            X 1
 0 X      A     X 0
-1 X            X-1
-2 X  X  X  X X-2
  -2, -1, 0, 1, 2
```

where X = a direction and A = the car. As can be seen, the car's direction is defined by two different variables: x for x-axis and y for y-axis. When the direction must be converted to a bigger or smaller gear, the shape of the square (obviously) stays the same. The sides just grow in length. We can therefore use a simple mathematical proportion to find the result.

Only one of the coordinates will be changed, as only one coordinate can change. If one of the coordinates is not either side length / 2 rounded down or its opposite number, we know we will be changing that coordinate. The other possibility is that neither coordinate fulfills the conditions. In that case we can choose either one, it will not make a difference.

The coordinate that is either side length / 2 rounded down or its opposite number will simply be set as the new gear's side length / 2 rounded down or its opposite number. The other, however, will be determined by a proportion. Let us look at a gear three changing to gear two.

The left one represents gear three, the right one gear two

```
X  3
X  2      X 2
X  1      X  1
X  0      X  0
X  -1     X -1
X  -2     X -2
X -3
```

We can now make the proportion:
x = x-axis in gear 1
y = y-axis in gear 1
h = (new gear's side length / 2 rounded down or its opposite number)
g = (old gear's side length / 2 rounded down or its opposite number)

The following statements must equal, because the sides are of equal length.
b / h = y / g
b = y * h / g
There we have b. The same can be done with a and x.

The most important algorithm of the program is in GearManager's newPosition-method, which must find the new location on the track while taking into account the possibility of crashing into an obstacle. This means it is not enough to just add direction into current position, but that every step must be moved one by one while at the same time mimic as closely as possible how the car would actually move.

First the intended destination is counted, so it is known where the car must finish if no obstacle is found. Then the car must be moved one square at a time towards it, unless the next square has an obstacle in it or the car has already reached its intended position.

The challenge is to go through the steps in an order resembling how the car moves in real life. It is of course easy if the angle the car drives in is divisible by 45 as all the steps then are similar: either x-axis or y-axis or both are advanced. How then to calculate the other possibilities?

For this program, the following way has been used:

b1 = number of original vertical or horizontal tiles, whichever is bigger
s1 = number of original vertical or horizontal tiles, whichever is smaller
b2 = number of remaining vertical or horizontal tiles, whichever is bigger
s2 = number of remaining vertical or horizontal tiles, whichever is smaller

The shorter direction is to be moved if s1 / b1 < s2 / b2. Otherwise the longer route will be taken. This way tile by tile progress will be made close to the way the car would move if it moved as in real life; the car will move as closely to a straight line between the two coordinates as possible.

6. Data Structures

The data structure mostly used is an array as it enables having an immutable size but mutable inside. Arrays of characters are used to represent the game track in all the different situations. And as the size of the track never changes but the cars driving on it do change places, the choice of an array was an obvious one. I do not see reason for using any other kind of a data structure, though it surely could have been implemented with buffers, for an example. It would have just been more inefficient and added the risk of accidentally changing the track's size.

Arrays were also used in the graphic design of the cars as they were an easy form to use for not only mirroring but also shifting the elements one index away from its place. The final forms were still saved as vectors because there was no need for changing them afterwards.

For storing lapTimes a buffer was used as it makes it easy to add an element to the end and check the last element of the structure.

7. Files and Internet access

The program uses two kinds of files: driver files and racetrack files. They are both .txt files and readable by people.

Racetrack files are formatted as follows:

Header: 12 characters
    text "FORMULATRACK": 12 characters

Record
- tag “REC”: 3 character
- holder
  - name length: 2 characters (numbers between 1 - 9)
  - name: (name length) characters
- time
  - time length: 1 character (number between 1 - 9)
  - time: (time length) characters

Track
- tag “TRACK”: 5 characters
- layout height: 2 characters (two numbers between 1 - 9)
- layout length: 2 characters (two numbers between 1 - 9)
- layout: (layout height * layout length) characters (different characters representing different tiles on the map)

Lap
- Tag “LAP”: 3 characters
- lap amount: 1 character (a number between 1 - 9)

End
- tag “END”: 3 characters

An example of a racetrack file can be found from the the folder /racetracks under the name of RaceTrackTest01.txt

Driver files are formatted as follows:

Name
- name length: 2 characters (numbers between 1 - 9)
- name: (name length) characters

Tracks driven
- track amount: 2 characters (numbers between 1 - 9)

Tracks
- text LAPTIMES: 8 characters
- tracks: track amount * Track

Track

track name length: 2 characters (two numbers between 1 - 9)
name: track name length characters
laps: 1 character (number between 1 - 9)
lap times: lap * 3 characters

An example of a driver file can be found from the the folder /drivers under the name of PetePetruska.txt

8. Testing

As I built the Game, RaceTrack, Car, GearManager and Gear, I tested each of them quite thoroughly, making a new test every time some feature was added and making sure everything worked well. Most of these tests can still be found in /tests directory. However, when I got them working pretty much as they are supposed to and managed to build a text based user interface for actually playing the game, my testing of the game quickly shifted into just trying the game out with the interface instead of coding separate tests. And so I continued testing with the interface instead of writing tests out when I built the graphic user interface.

Some of the tests:

Class Car was tested by setting it up into a certain position and then making it drive with certain parameters in a premade track. Then it was checked whether the car's position was as it should be if program worked as it is supposed to. For example just driving straight, or trying to drive through a wall or another car. This would also test whether the GearManager found the correct position. The seekPossibilities-method was also tested a little by comparing the returned array to the correct one.

GearManager was thoroughly tested. newPosition-method was given values and seen if it returned the correct position. Changing gears was tested by setting up pre chosen direction, then switching gears and seeing if the new direction matched to the intended one. Upping and lowering the gear was also tested.

RaceTrack only had few tests to see if it would draw the track correctly from the given string and then draw the cards and their tracks in the correct positions with drawTrack. These were both tested by comparing them to the correct outcome track.

Changing direction was tested by many, many different scenarios and special situations. Different tests had to be made for corners and sides and plenty of different tests had to be ran inside those groupings because there were several possible scenarios depending on the

side or corner, negative or positive values. It was simple to look whether the direction arrays matched with the intended arrays.

With the interfaces, the testing mostly consisted of driving and looking whether the tiles looked as they were supposed to and if the files were read and saved in the right form afterwards. I've also tested that the application does not crash if the possible destinations go overboard.

Sadly I have not had the time to make sure all the old tests still work; some of them do, some of them don't.

9. Known bugs and missing features

The biggest bug of the game, I believe, is that it does not let the user choose a wrong kind of a file, it has no mechanism to make sure the file information it reads is formatted correctly. It would best that in the loops in FormulaGUI where the files are chosen, they would carefully checked for proper formatting (down to numbers being in their correct places, not just the headings) and another required if any fault were to be found. At the current form the program simply crashes if a wrongly formatted file is chosen, often without even giving out any specific information about the reason. Driver files have no quality control at all while the racetrack files are checked only for the correct headers by a Game object, stopping the program by throwing an exception with information about what exactly is wrongly formatted.

A missing feature all over may have been putting in way too few possible exception throws to make sure everything works fine. I have, however, done my best to make sure in the program design that nothing that can crash the program gets done. Besides the files, there is one glaring, known exception to this rule: if the car tries to drive over the track's boundaries, the game crashes as an invalid index number is tried. But I do not think this is a huge problem, as one can reasonably in my opinion assume every track is walled off on the sides. Still, it would be nice to fix so it wouldn't need to be so. The possible future destinations going overboard do not crash the program, but some otherwise useless buffer walls around the map are still nice, as it allows the player to better the car's direction.

To my knowledge, those are the only two ways the program may crash, and there should be none as all other user input is filtered through buttons that only give out good values.

A small problem I've noticed is that as the car drives tile by tile according to the afore explained algorithm, this creates a situation where the car acts a little bit differently depending which way it is driving. Often, this is not a problem at all, but when trying to avoid hitting a wall by a narrow margin, it is hard to remember which precise route the car takes

when planning a turn or two in advance. The current turn is no problem of course: one always sees precisely where the chosen moves lead.

Also it is currently possible for players to cheat if the finish line can be reached by driving the track backwards as there is nothing to make sure the player has to go round the entire track. This problem can certainly be eliminated by making the track a one way road only (as has been done in the provided test track) but this means the feature of several laps is not a possibility. One can also always trust in player honesty, but it is quite an annoying thing.

10. 3 best sides and 3 weaknesses

Many of the weaknesses were already raised in the previous part, but here are 3 more:

1. While the files can be read with the knowledge of the formatting at hand, the driver files are not at all readable without the format at hand, as they have very few markers for the numbers. Racetrack files suffer some of the same problems, but I think they are still quite readable even without a direct reference at hand every single moment.

2. I may have made the class structure somewhat unoptimal, not having strong enough interfaces with each object, having some things be too interconnected. This leads to situations where FormulaGUI must reach through Game to Car to GearManager to get a piece of information (lap times). This is not exactly optimal planning. One might consider is it actually necessary to have a separate GearManager, for an example, or have direction in Gear instead of GearManager. The original choice of creating the GearManager was made so that Car would be a clearer class. It also seems strange and unoptimal to RaceTrack keeping track of the physical tracks and GearManager and Car drawing the possibility tracks over it, but the information from GearManager and Gear are needed, so I am not sure how I would have otherwise implemented it. Maybe merge Car and GearManager and give direct access to RaceTrack for Car?

3. I also think that all of the implementation is not made in clearest possible way which makes understanding what it says more difficult. I should have made those parts easier to read from the beginning and I did plan to come back to them but never had the time.

1. A good side, I think, is the system showing the player future destinations. The ignoring of obstacles in the general view makes sure the player has a clear understanding of what his/her car’s current direction is, while the single destination view makes certain that the precise choice is clear. I also think that in general, the car driving as close to the physical line as possible is good, the method was just not quite perfected.

11. Deviations from the plan, realized process and schedule

I did deviate from my original plan in the sense that I first developed RaceTrack and all the classes associated with car to have basic functionality before making Game (though I did make the file format reader from the very beginning) and then worked on them in a tandem for about two weeks before switching to first making the text based user interface, developing the game some more, and then spending two and a half weeks working on the FormulaGUI and of course also making some changes in to the game's internal logic. But mostly I worked on the FormulaGUI.

All in all, the game's internal logic took somewhat longer than I had anticipated, but the graphic user interface really took a lot more time.

12. Final evaluation

I think the program is successful in doing what it does. It has some problems as documented above (the files, no multiple laps etc.), some even big, but it does still work fine and well. If the files are correct, the game works: You can have a race and the program mimics how a car drives. All the information about where your driving will take you this turn is readily available so you won't make bad moves because you were under the wrong impression.

While looking at the drivers files is not currently something a user would probably think to do, the information is there. Had I had more time, I would have added a way to read the files in a nice format through the user interface. I should have also added more than one best result to the track file; a top five list at least would have been nice, but an even longer one would not have hurt either.

I think the structure of the program is not as great for changes and extensions as it could be, but I believe it still works somewhat well at least. It has some clear units with small interfaces and all the longer referencing is still done through "correct" lines.

Were I to start the project from beginning, I would at least get rid of GearManager as a separate class and merge it instead with Car. I would also at the very beginning put the functionalities where they should be and make the most sense. For an example, reading the file information in the Game object should never have taken place. I would also create more submethods inside some very long methods in order to make the methods clearer to read. GearManager.newPosition really suffers from this. It would be much clearer to read with parts of it in well named submethods.

13. References

Besides the course material and Ohjelmointi 1 -course's materials I have used the following materials:

Several threads in Stack Overflow when searching answers for small problems:
https://stackoverflow.com/

This site mostly for GUI Programming tutorial: http://otfried.org/scala/gui.html

Java swing tutorials and documentations:
https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html
https://docs.oracle.com/javase/7/docs/api/javax/swing/JFileChooser.html

Several different pages of Alvin Alexander:
https://alvinalexander.com/scala/fp-book/how-to-use-by-name-parameters-scala-functions/

14. Appendixes

Source code will be provided on the side as an Eclipse project.