Formula game

Mikko Juusti

793469

2020

27.4.2021

General description

The formula game is a turn-based racing game where one to two players are competing against each other or against the clock in a practice setting (if only one player and no bots). The formulas have 5 gears in total, and they can be changed max one step up and down per turn. With higher gears the pace will get higher, but controllability of car will be harder as turning is harder. Each turn players are also capable of changing the direction one step to the left or the right or keep the original distance. If two cars tried to go to the same point, one car would cut in and the other would not be able to be in the same point. The player first to cross the goal line during the last round, will be the winner. If a player goes out of track, the player will have to skip 3 turns and back to gear 1 with direction turned 180°. Fastest laps are saved to the scoreboard.

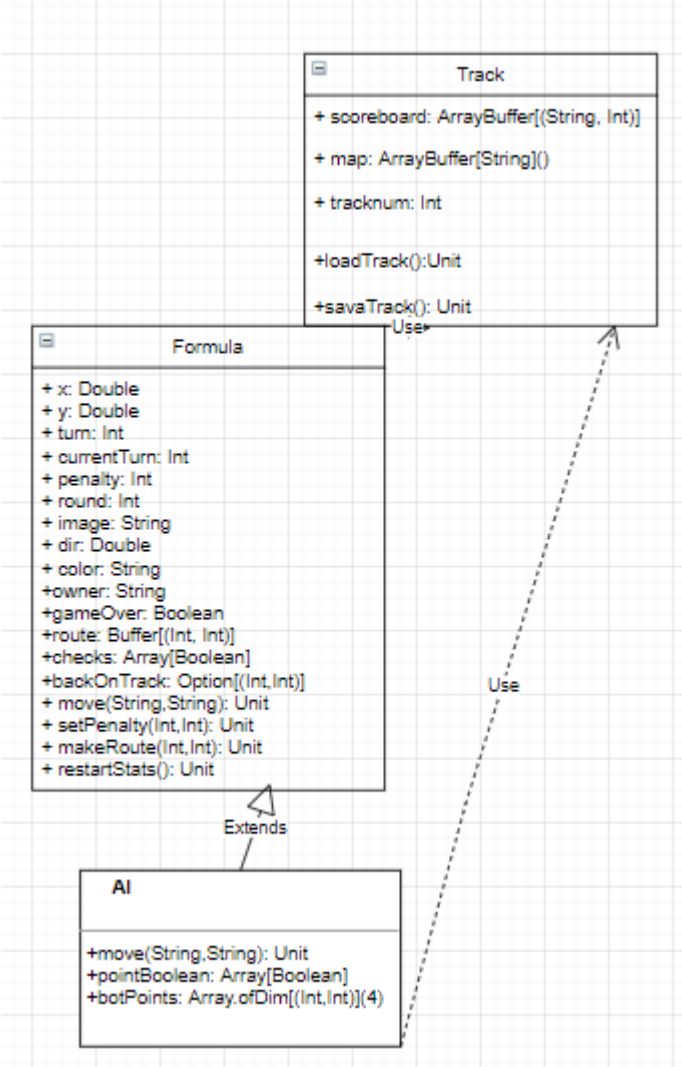
User interface

I decided to make the user interface entirely graphical as I thought the user would get better experience that way. All user actions but typing player names are done by mouse clicks. When starting the formula game, a main menu opens. There the user can select player amount and type (AI or human player). Also, human players get to choose names for them that are visible in game. The user also selects number of rounds and the track where the game is played on. The user can only select tracks that are available. After those options, the user is ready to press play game button that changes scene to the game.

In the middle of game scene there is the track where the formulas are in. The formulas are controlled with the toggle buttons on the right side. When the user has chosen possible direction and gear changes, then using move player button the formula moves. Below the buttons there are info about the player in turn. On the left

side there is a scoreboard of the fastest laps, small help text to moving formula, new game button and restart game button. The restart game restarts the game using the options previously selected and the new game button goes back to main menu scene to rechoose the options.

Program structure



Contrary to my plan I got rid of my Player class and block class as they had not much in them, so I united formula with player and used blocks directly in the GUI app. When doing the AI I felt like extending formula was the correct way as moving would be the only change between it and formula class. The move method is overridden, and I have private method check that is close to original moving method.

I did not include the GUI app in the UML above as it is way too big for to show that way. That is the biggest downside of this project that I did not separate GUI in the smaller classes. When doing the GUI, I realized that there were a lot of dependencies in the GUI objects, and I did not find ways

to refactor it so that dependencies stay the same. In the GUI I have two scenes menu and game stage. The user's selections transfer from menu to the game. The GUI app loads selected track and from the track are seen the players/AIs starting points. I used border pane to get benefit from entire screen and to place GUI items as I wanted. The track was represented by grid pane, so I was able to use the x and y coordinates. It was also easy to place the track blocks with them. I also used Vboxes in the sides of the game and also in main menu.

Algorithms

The main algorithms in this game revolve around formulas movement. Firstly, I made moving based on the user's input. Based on the gear, the direction changes by ± 2 * Pi / (8 * gear) depending on turning choice. So, with bigger gear it gets harder to turn. With direction and gear known it is possible to get the new x and y coordinates with the formulas under.

y += gear * (-sin(dir)), x += gear * cos(dir)

After that it was possible to move correctly but if formula would cut through a corner, it could not be seen without knowing the formulas route. So, I made a makeRoute function to get every grid that the formula goes through. To help achieving this, I made a line function that returns y value depending on x value and iterated it in for loop to get all the possible pairs. When $\Delta x=0$ I had to use for loop with constant x and iterate over the possible y values. With route it was easy to notice when crash or cutting scenarios happen.

def line(newx: Double) = ((oldY - y.round.toInt) / (oldX - x.round.toInt)) * (newx - oldX) + oldY

When it came to the AI's moving algorithm, I had two implementations in mind. First, I tried version that scans through all the possible moves and then decides from them. It was not efficient as it checked in worst cases 9 different possibilities. After that I went for the current implementation with helping points for bot. When getting close to the point bot knows the next direction where it should be heading. When too close to inner wall it turns outwards and when too far away it turns inwards. The ideal place for bot is 2 grids from the inner border. The bot also avoids too big of an angle with the heading direction as it takes many rounds to straighten the formula.

Data structures

In this project I used a lot of different data structures. I found it useful to use Map with formulas as I needed the player number a lot. I also used Array.ofDim for the checkpoints and bots helping points as there are always the same amount of them and it was easy to set them for specific spot. With some cases I used Option as it is safe way to handle situations and also None can be used as a starting value. For other cases I used buffers and arrays for their easy-to-use and lists for Scalafx-related tasks. For track I used Array of strings so I could use for loops to get every character and place them accordingly.

Files

For each track there is an own file where there are starting and ending headers for track and scoreboard. Inside track headers there are characters where each represents something in gui. ‘#’ is out of track block, ‘*’ is a track block and ‘1’ and ‘2’ are players starting blocks. ‘G’, ’R’, ‘D and ‘L’ are checkpoints that forces players to go clockwise and not cheat by just turning 180 degrees and then going through goal line. ‘Z’, ’X’, ‘C and ‘V’ are bots helping points that are positioned in the corners 2 steps from both sides. The last 8 characters need to be in the said order when setting them clockwise. In scoreboard position is separated with ‘.’. Name and turn count are separated with ‘:’. All in all, the example file looks like the one below.

```
/TRACKNR1
##############################################
##############################################
########************************##############
########************************##############
#######***********2************#############
####**************1**************###########
##*******V****************Z*******########
##**********************************########
##**********######G#######*********#########
##**********#############*********#########
##**********#############*********#########
##**********L############R*********#########
##**********#############*********#########
##**********#############*********#########
##**********######D#######*********#########
##**********************************#######
##********C****************X*********#######
##**********************************#######
##**********************************#######
##**********************************#######
###*********************************#######
##############################################
/TRACKRDY

/SCOREBOARD
1. MOCCO: 23
2. MOCCO: 23
3. Player 1: 23
```

```
4. Player 1: 24
5. Player 1: 24
6. MOCCO: 26
7. MOCCO: 28
8. ENEMY-BOT: 30
9. ENEMY-BOT: 30
10. ENEMY-BOT: 31
/SCORERDY

/ENDTRACK
```

I made methods for loading and saving the track in Track class. The save track is used when there is an update in scoreboard. The game also loads images and sounds from files. I made a function for playing music.

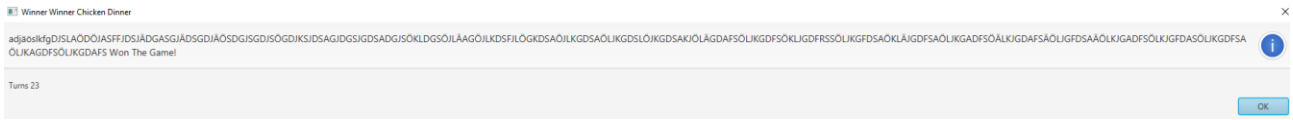
Testing and bugs

The testing went kind of how I intended. First, I spent I a lot of time getting to the bottom of my moving algorithms and implemented the idea then transferred it to Scala and tested how it worked. As I got the basic moving working, I tested file loading until it became bug free. After those I was able to move towards testing with GUI. I used a lot of printing for example in making the route function. When my GUI was on a better form, I was able to see if crash and cutting worked as intended. One bug I saw what was unexpected was when I added rotation for formula. Rotating image came out of the grid spot and left pieces of formula on the road. I decided to crop the image to circle shape to overcome this issue. I knew that when going out of the grid, it would break the game so I that was one thing I focused on. (It can be tested with track number 2.) All in all, seeing the bugs made it easier to understand if something was missing.

Towards the end when creating the main menu, I did a lot of testing to see that the information was smoothly transitioning between two scenes and found out bugs when doing so. For example, the restart button was not working anymore as I added more complexity to the program, but it was fixed by creating a restartStats method. I also tested the game with two different 1080p screens (15” and 24”) and to my surprise the game did not scale the same even with same resolution. That is why I used screen size as well in the GUI code. Sadly, I had no other resolution devices to check how the game looks like on a better resolution. So, there might be still room for improvement.

After all the testing I have done I feel like the game is stable and now I have not found any new unresolved bugs. The user can only give the input that is needed and nothing else. One thing that comes in mind is when giving way too big of a name the game will not be able to show it entirely. But even with long names the game can load and save files like it should. The victory message is

still able to show the name, so I feel like it is a minor bug. But it could be capped to certain number of characters for visual improvement.



3 best sides and 3 weaknesses

The game's best side is the stability I have reached with a lot of testing. Always when I found a bug, I used time to get to the bottom of the issue and then solve it instead of creating something new. The game has all the requirements and I feel like I did not make any compromises on testing to reach it.

I also feel like my algorithms are key to this project. The formula is moving as intended and all the edge cases have been noticed. When cutting through a corner the formula will crash as the route is taken care of. I am also proud of the AI as it can go track multiply rounds without crashing. Only way to crash the car has been to cut with another formula but that was also how I wanted it to work.

Even though this was my first GUI I ever made, I still feel like the user experience is great at least on a functional side. All the main functionalities work together, and I feel like the game is clear for the user.

On the other hand, as this was my first GUI, the visual side could still be improved. I feel like there could be visual improvements here and there as I mainly used basic elements. It was also hard to make small grid slots look decent.

As the biggest weakness in this project, I see that my GUI object is way too big. I had hard time figuring out how to separate it to a smaller part. There was a lot of dependencies between different scenes and GUI-objects, so the refactoring was not so simple. I feel like the approach I ended up came with inexperience with the GUI. Refactoring and cleaning the code would be the next thing I would do if I ended up continuing this project.

Even though I feel like the gameplay is on a good state I am aware of that the track structure is simple. With current code it is only possible to make tracks with rectangle shape and both checkpoints and bot helping points requires that there exist 4 each. So, on behalf of that the game is lacking expandability as it would need some new code to implement even though it is not an impossible with this codebase.

Deviations from the plan, realized process and schedule

When starting I quickly realized that I would like to start making the GUI as fast as possible as I had no experience on how long it would take. So, after getting the basic functionalities on file reading and car movement I started with it. My choice was not to make the text version first and go straight for the GUI as I aimed for the challenging difficulty. After that I started implementing the needed features by doing the easiest first. On the second progress check, the game was almost checking all the intermediate requirements and on the third I also got the AI working. I feel like my schedule held well as I thought I reach the state of stress-free coding as I started early. I also liked a lot of the process of making the game, so it got me motivated.

Final evaluation

Overall, I feel like I reached my goals with this project as I got all the requirements done. The game is stable and works well with different scenarios. In the beginning the project felt like a mountain too high to climb as this was my first project from start to finish. But I felt like that was the right mindset as I took this as a serious challenge and started working early on. I feel like I made a great playing experience for the user as user cannot accidently break the game. The biggest downside of the project is in the codebase invisible from the user. My main app object is way too big, so it is not optimal for reading and working with the code. If I made this project again, I would make the class structure better for the GUI side. It was hard to know how GUI would be separated and after a while it was too full of dependencies so splitting the object was hard. The number of players is easily increased if needed but changing the track structure from the simple rectangle shape would need some extra effort. I feel like the algorithms in this game are a big upside of this project as with them the game is working well, and they handle a lot of different scenarios.

References

http://www.scalafx.org/

Scalafx playlist

Appendixes


Formula game
THE FORMULA GAME
Select Players
2 players
Select Player 1 name
Mikko
Select Player 2 name
Player 2
Select round amount
1
Select track
1
Play Game


Formula game
New Game
Restart Game
Fastest laps
(MOCCO,23)
(MOCCO,23)
(Player 1,23)
(Player 1,24)
(Player 1,24)
(MOCCO,26)
(MOCCO,28)
(ENEMY-BOT,30)
(ENEMY-BOT,30)
(ENEMY-BOT,31)
Move by selecting gear and direction. Then press Move Player.
Gear Up
Gear Down
Turn Left
Turn Right
Move Player
Player: Player 2
Turn 0
Gear 1/5 (Blue)