

## Formula 0: Project Document

Ian Masila | 100278020

Mathematical, Computational, and Statistical Sciences | Year III

05 May 2022

## General description

*Formula 0* is a two-player formula racing game. The game features a *text-based user interface* where two race car drivers race around a racetrack. The following rules have been implemented into the latest build of the game:

### Game Rules

- The game area is  $[n \times m]$  sized grid on which the racetrack is drawn, where  $n$  and  $m$  are natural numbers.
- The car has five gears. The velocity and turning circle are dependent on the gear.
  - The gear can be changed by one step on every move.
  - The direction of the car can be altered by one step on every move, or one can keep going in the same direction.
  - When changing gears, the “same direction” is the direction which is closest to the direction of the last move. There are different variations of this game, but in *Formula 0*, when changing gears and/or direction, we must first calculate the “same direction” and the turning directions are presented based on this direction.
- The players move their cars in taking turns according to the movement rules described above:
  - On each turn, a player may change gear, one step up or down, or make no change.
  - In addition to this, the player may turn the car one step to the right or to the left or continue forward.
  - When changing gears, the “same direction” is the direction, which is closest to the direction of the last move.
- Both cars start the race facing in the direction of the track, both set to gear one.
- A car that has gone out of the track is considered off the road (even with larger gears one is not allowed to go off track between two points).
  - *A car that is off the road is reset to its previous valid track position. Note, however, that any direction and gear changes remain in effect.*
- Two cars are not allowed to be in the same point (enables one to cut in).
- The players’ starting points are randomised on the starting line.
- The winner is the one first to cross the goal line. A winner is chosen at random in the event of a simultaneous goal line crossing.

In my opinion, the latest build of *Formula 0* exhibits a moderate level of implementation difficulty. Though text-based interfacing may seem simple, its implementation is anything but due to the low level of abstraction involved as compared to graphical visualisers.

## User interface

*Formula 0* exists as a package of program files comprising a “Main” object and several class files modelling aspects of the game.

### *How is it started?*

To launch the game, run “Main.scala”. A loading screen will appear on the terminal followed by the main menu screen where the user is prompted to key in “1” or “0” to play or exit the game respectively. If the user chooses to play the game, player information such as name and car character is fetched from both players.

A text file containing track information, “Tracks.txt”, is then read and its lines are printed on-screen. A track is represented as an array of strings separated by a new line character. The user is then prompted to key in the name of the track they desire to play. Once a track is selected, the initial game state is *loaded*. Loading involves randomly placing both race cars on the starting line of the track, displaying the track on-screen, and setting up turn-based gameplay. Racing can now begin!

### *What can you do in Formula 0?*

At each game state, a single player controls their own car. The current player is prompted to shift gears (“S” for shift down, “0” for no shift, or “W” for shift up) then move in a certain direction (“A” for left, “W” for forward, or “D” for right). The user selection is advised by car status information displayed below the track. Most of the relevant game information (turn status, car status, lap status) is presented on-screen below the track therefore the player only needs to focus on racing by keying in the desired action.

## Program structure

*Formula 0* has a modular structure with each module performing a specialised task. The final realized class structure is as follows: class *Grid* modelling the racetrack, class *Player* modelling the player, class *RaceCar* modelling the race car using a state design pattern, and class *Formula0* modelling the game as a transition state system and describing the user interface. There is also a *Main* object for consolidating all modules and running the game.

Class: Grid

```
Methods:
setGrid(track: Array[String], wall_tile: Set[Char] = Set('X'), starting_line: Char = '#'): Unit
gameCrashScreen(crashChar: Char): Unit
toString: String
```

Class: Player

```
Methods:
toString: String
```

Class: RaceCar

```
Methods:
shiftUp(): Unit
shiftDown(): Unit
noShift(): Unit
right(): Unit
left(): Unit
forward(): Unit
toString: String
```

Class: Formula0

```
Methods:
getTurn: Int
getCurrentPlayer: Int
getPlayerNames: Array[String]
getPlayerFromName: Map[String, Player]
getCarFromPlayer: Map[Player, RaceCar]
getLap: Map[String, Int]
getNextState: Formula0
successorState: Formula0
isWinner: Array[String]
deepcopy(): Formula0
toString: String
```

Class: Game

```
Methods:
gameLoadingScreen()
gamePlaySelectScreen(): Unit
gameTrackSelectScreen(): Unit
gameWinnerScreen(name: String): Unit
gameResultsScreen(results: Array[String]): Unit
gameOverScreen(): Unit
playSelect(): Unit
carCharSelect(): Char
car1CharSelect(car0Char: Char): Char
trackSelect(): Tuple2[String, Array[String]]
lapSelect(): Int
load(): Formula0
play(): Unit
```

There are several relationships between these classes, most of which are association and dependency relationships given that Formula 0 is a game. For instance, the game class has an association and dependency to the race car class.

## Algorithms

The main algorithms involved in the running of Formula 0 address the movement of the race car. A car has a gear state, a direction it is facing, a set of applicable directions (left, forward, and right) in which it can move, and a position on the track. To handle changes to these attributes through the course of a game, several algorithms are invoked. Note that this requires viewing the track grid as a vector space with the origin (0,0) as the bottom-left corner of the grid and directions as vectors themselves. (Vectors are implemented by class *Vector2D* of *Formula 0*.)

To shift gears, the car's direction vector must change appropriately since different gears allow different direction vectors to be possible. The same applied for changing direction. To find the correct direction vector for the car given a certain gear/ direction change, the "same direction" must be calculated. This "same direction" is the direction with the least deviation from the original direction, making for the most realistic shift in car direction. To calculate this "same direction", a set of candidate directions is algorithmically inferred after which the arcus tan of each candidate is compared to the arcus tan of the original direction vector. The candidate direction with the least absolute difference with the original direction vector is the "same direction". This becomes the car's updated direction after a gear/ direction change.

Candidate directions are inferred algorithmically by using spatial geometry and basic vector calculus. The same applies to updating applicable directions after updating the car's direction vector. To update the car's position after a gear/ direction change, vector addition of the original car position and the direction vector of the change is sufficient.

*How does the program move the car on-screen?*

Since the terminal only prints strings, the track grid is represented as a long string with new line characters at the end of each row. To find the string index corresponding to the car position, the following formula applies:

```
nextState.carPosGridView = (gridViewLen - 1) - ((car.carPosition.x * (track(0).length + 1)) + (track(0).length - car.carPosition.y))
```

where *nextState* is the game's state after the current player plays, *car* is the current player's car, and *track* is an array of strings representing the track grid.

A game state can then be visualised by printing the editing the original track string to include the car characters at the appropriate indices.

## Data structures

Scala's basic data structures are used to process data. They include arrays, buffers, sets, maps, and tuples. These structures were chosen due to them being lightweight and quite easy to use for the game's requirements. More complex data structures such as binary trees could have been used to model the adversarial gameplay since they lend themselves to decision process algorithms such as minimax and pruning. However, these structures were not necessary given the simplicity of *Formula 0*.

All these data structures are immutable except for buffers and maps whose states needed updating. Moreover, I defined my own data structure for handling vectors, `Vector2D`. It models two-dimensional vectors:

```
case class Vector2D(x: Int, y: Int){  
  def magnitude = scala.math.sqrt(x*x + y*y)  
  def +(that: Vector2D): Vector2D = Vector2D(x+that.x, y+that.y)  
  def -(that: Vector2D): Vector2D = Vector2D(x-that.x, y-that.y)  
  def *(scalar: Int): Vector2D = Vector2D(x*scalar, y*scalar)  
  def dot(that: Vector2D): Int = {x*that.x + y*that.y}  
  def equals(that: Vector2D): Boolean = (x == that.x) && (y == that.y)  
  def toTuple = Tuple2(x, y)  
  // 2-dimensional vector defining row, column coordinates  
  def toRowCol = Vector2D(y, x)  
  def deepcopy() = Vector2D(x, y)  
}
```

Note the method `toRowCol`: the track grid consists of rows and columns which correspond to the y and x axis respectively. Note also the `deepcopy()` method which returns a deep copy of a vector. This is especially important in the game to avoid unwanted dependencies.

## Files and Internet access

There are two text files that *Formula 0* deals with: Tracks.txt and GameHistory.txt. In Tracks.txt, track information is represented with keyword tags describing attributes such as format, name, fastest lap, and fastest driver. The tracks themselves are delimited by “...” with each row written on its own line and an “ENDOFTRACK” tag to signal the end of a track. This format is crucial for the parser to read track information properly. Here is an example of a proper tracks file:

```
format: only track rows and ENDOFTRACK tags should be enclosed with "...", finishLine is
always '#'
name: defaultTrack
track:
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXX          XXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXX          XXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXX  XXXXX  XXXXXXXXXXXXXXX"
"XXXXXXXXXX  XXXXXXXXXXXXXXX  XXXXXXXX"
"XXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXX"
"XXXXXX  XXXXXXXXXXXXXXXXXXXXXXXX  XXXXX"
"XXXXXX  XXXXXXXXXXXXXXXXXXXXXXXX  XXXXX"
"XXXXXX  XXXXXXXXXXXXXXXXXXXXXXXX  XXXXX"
"XXXXXX  XXXXXXXXXXXXXXXXXXXXXXXX  XXXXX"
"XXXXXX  XXXXXXXXXXXXXXXXXXXXXXXX  XXXXX"
"XXXXXXXXXX  XXXXXXXXXXXXXXX  XXXXXXXX"
"XXXXXXXXXXXX  XXXXXXXXXXXXXXX  XXXXXXXX"
"XXXXXXXXXXXX  XXXXXXXXXXXXXXX  XXXXXXXX"
"XXXXXXXXXXXX  XXXXXXX  XXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXX  XXX  XXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXXX  X  XXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
"ENDOFTRACK"
fastestLap(seconds): 0
fastestDriver: Anonymous
```

GameHistory.txt is the text file onto which a finished game's results are written automatically. This information is as follows:

```
"id":1234567890,
"date":"26042022",
"trackName":"defaultTrack",
"winner":"Anonymous",
"loser":"AnonymousOther",
"winnerCarName":"Anonymous",
"winnerCarCharacter":"@",
"winnerLapTimes":[120, 100, 80],
"winnerBestLapTime":80,
"loserLapTimes":[140, 133, 100],
```



```
"loserBestLapTime":100,  
"winnerWallCrashes": 3,  
"LoserWallCrashes": 11,  
"winnerCarCollisions": 1,  
"loserCarCollisions": 4,  
"runTime": 673
```

## Testing

Even though I had planned to take a test-driven development approach, I found it most efficient and effective to test the game by playing the game. When undesired gameplay occurred, I would observe what happened, contrast that to what was supposed to happen, and then infer the culprit.

## Known bugs and missing features

When a player crashes into the wall or into the other player, the game resets the car to the previous position with any gear/ direction changes remaining in effect. This was a deliberate decision partly to make the game more challenging by penalising crashes. However, another alternative would have been to reset the car to the previous state, i.e., reset position, direction, as well as gear state. To do this, I would apply the inverse transformation on the car. For instance, if a player shifted up, turned left, and crashed, I would turn the car right and shift down to reset the direction and gear. However, after some iterations of this, there would be a mismatch between the gear and the applicable directions of the car. This is the main known bug.

Another bug in the game is that the cars may be randomised onto the same spot of the starting line at the start of the game, though rarely. The only effect is that the second player's car will be invisible momentarily until the first player moves their car. I left this bug in as a quirk of *Formula 0*.

Missing features: the game cannot print the fastest lap time and fastest driver information onto the tracks file. This information must be recorded manually for now.

### 3 best sides and 3 weaknesses

One of the aspects of *Formula 0* I am most proud of is the fact that the track grid is a vector space and a string at the same time. This made implementation that much more complicated as one must translate between vectors and string indices in a precise manner to ensure the game runs smoothly. Nonetheless, it is amazing how complex manoeuvres can be displayed on the terminal and make for a thrilling game.

Moreover, I think it is incredible that gameplay can be easily customised by any user by editing the tracks file. I must admit that it is one weakness that the default track is quite difficult to manoeuvre around. It would have been better to make an easier default track for users to make the game's learning curve gentler. However, this weakness can be circumvented by users making their own tracks, adding them to *Tracks.txt*, and selecting them on the game's loading screen.

The last strength I find is the game's adaptability. The source code provides plenty of opportunities for updating the game with ease. This should greatly benefit the evolution of the game.

The main weakness with *Formula 0* is the main bug mentioned previously. I would prefer if the car's entire state were reset after a crash rather than just its location. I have commented out such an implementation because it does not behave consistently for yet unknown reasons. However, some players may prefer that only the car's position is reset and not the gear state nor the direction vector.

The last weakness with the game is that it is skewed towards penalising the player for wrong moves. If I had more time, I would also implement a reward system in the game such as collecting coins or power ups such as immunity from crash resetting.

### Deviations from the plan, realized process and schedule

There were major deviations from the plan of two-week sprints. Unfortunately, I could not balance my workload effectively to stick to the plan. I had to prioritise other assignments that were more time sensitive and once those were done, I could then focus on the project. I ended up doing the project in effectively two and a half weeks of concentrated attention. The project progressed rather slowly initially as I tried to get to grips with the car movements and how to display the game on the terminal. Then progress was quite erratic as I had breakthroughs interleaved with dead ends. Progress then slowed down in the final stages of the project as I tried to finetune the game to my liking. Ultimately, my plan's time estimate did not at all match with reality. I realised that my plan was too idealistic, and that reality is more chaotic. I learnt to anticipate such chaos and take it into account.

## **Final evaluation**

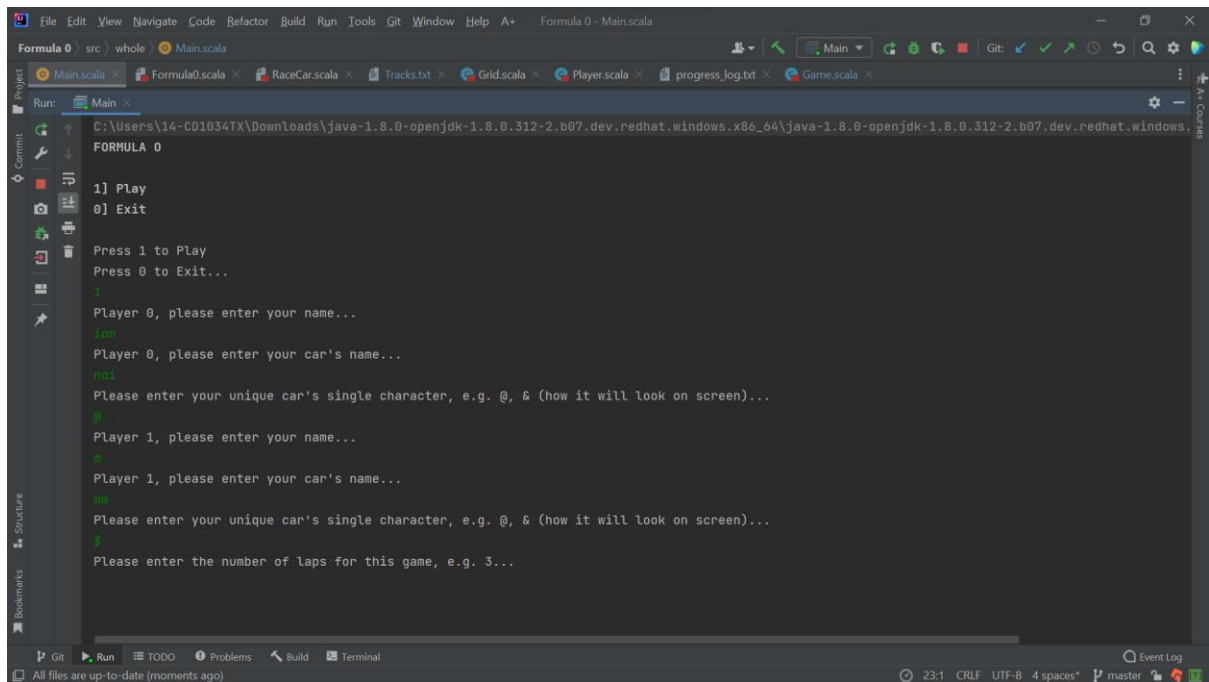
I am pleased with the final program. Considering it being my very first game/ project, I am happy that I was able to make a game that people can play and recreate as they wish. I think the overall quality is better than good despite some of the shortcomings previously mentioned. I deem it so because the game is not fixed but can easily evolve since the players can be creators themselves. For example, players may create a track with shortcuts if they so wish.

With the game's propensity for penalising wrong moves, a reward system could be implemented to make the game more balanced. With the scaffolding offered by the game's structure, making necessary changes should be simple and worthwhile. If I restarted the project, I would have kept in mind the user experience and not only focused on the functionality of the game.

## **References**

Scala API Documentation: <https://docs.scala-lang.org/api/all.html>

## Appendixes



```
Formula 0 - Main.scala
src > whole > Main.scala
Run: Main
C:\Users\14-CD1034TX\Downloads\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe
FORMULA 0

1] Play
0] Exit

Press 1 to Play
Press 0 to Exit...
1

Player 0, please enter your name...
ian

Player 0, please enter your car's name...
red

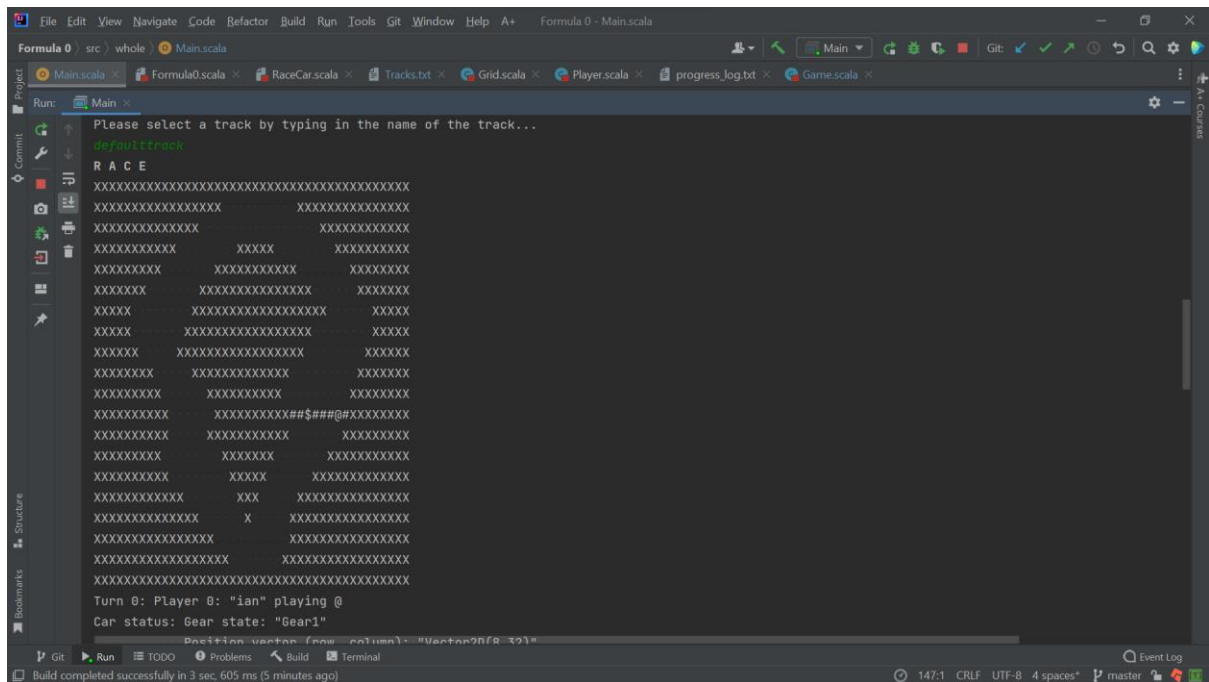
Please enter your unique car's single character, e.g. @, & (how it will look on screen)...
@

Player 1, please enter your name...
m

Player 1, please enter your car's name...
blue

Please enter your unique car's single character, e.g. @, & (how it will look on screen)...
$

Please enter the number of laps for this game, e.g. 3...
```



```
Formula 0 - Main.scala
src > whole > Main.scala
Run: Main
Please select a track by typing in the name of the track...
defaulttrack

R A C E

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX
XXXXXXXXXXXX  XXXXX  XXXXXXXXXXXXXXX
XXXXXXXXXXXX  XXXXXXXXXXXXXXX  XXXXXXX
XXXXX  XXXXXXXXXXXXXXXXXXXXXXX  XXXXX
XXXXX  XXXXXXXXXXXXXXXXXXXXXXX  XXXXX
XXXXX  XXXXXXXXXXXXXXXXXXXXXXX  XXXXX
XXXXXX  XXXXXXXXXXXXXXXXXXXXXXX  XXXXX
XXXXXX  XXXXXXXXXXXXXXX  XXXXXXX
XXXXXXXXXXXX  XXXXXXXXXXXXXXX  XXXXXXX
XXXXXXXXXXXX  XXXXXXX  XXXXXXXXXXXXXXX
XXXXXXXXXXXX  XXXXXXX  XXXXXXXXXXXXXXX
XXXXXXXXXXXX  XXXXX  XXXXXXXXXXXXXXX
XXXXXXXXXXXX  XXX  XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX  X  XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Turn 0: Player 0: "ian" playing @
Car status: Gear state: "Gear1"

D:\14-CD1034TX\Downloads\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe
Build completed successfully in 3 sec. 605 ms (5 minutes ago)
```

```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help A+ Formula 0 - Main.scala
Formula 0 src whole Main.scala
Main.scala Formula0.scala RaceCar.scala Tracks.txt Grid.scala Player.scala progress_log.txt Game.scala
Run: Main
XXXXXXXXXXXXXXXXXXXX XXXXX
XXXXXXXXXXXXXXXXXXXX XXXXX
XXXXXXXXXXXXXXXXXXXX @ XXXXXXX
XXXXXXXXXXXXXXXXXXXX XXXXXXX
XXXXXXXXXXXXXXXXXXXX XXXXXXX##$###XXXXXXXXX
XXXXXXXXXXXXXXXXXXXX XXXXXXX XXXXXXX
XXXXXXXXXXXX XXXXXXX XXXXXXX
XXXXXXXXXXXX XXXX XXXXXXX
XXXXXXXXXXXX XXX XXXXXXX
XXXXXXXXXXXX X XXXXXXX
XXXXXXXXXXXX XXXXXXX
XXXXXXXXXXXX XXXXXXX
XXXXXXXXXXXX XXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Turn 1: Player 1: "m" playing $
Car status: Gear state: "Gear1"
Position vector (row, column): "Vector2D(8,28)"
Direction vector (x, y): "Vector2D(0,1)"
Applicable direction vectors (x, y): "Vector2D(-1,1), Vector2D(0,1), Vector2D(1,1)"
Lap status: (ian -> 0, m -> 0)
Please select a gear shift...
[W] ShiftUp - [S] ShiftDown - [0] NoShift
Please select a direction to move...
[A] Left - [W] Forward - [D] right
Build completed successfully in 3 sec, 605 ms (5 minutes ago)
```

```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help A+ Formula 0 - Main.scala
Formula 0 src whole Main.scala
Main.scala Formula0.scala RaceCar.scala Tracks.txt Grid.scala Player.scala progress_log.txt Game.scala
Run: Main
XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXX
XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXX
XXXXXX XXXXXXXXXXXXXXX XXXXX
XXXXXX XXXXXXXXXXXXXXX XXXXX
XXXXXX XXXXXXXXXXXXXXX XXXXX
XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXX
XXXXXXXXXXXX XXXXXXXXXXXX $ @ XXXXXXX
XXXXXXXXXXXX XXXXXXX XXXXXXX
XXXXXXXXXXXX XXXXXXX###XXXXXXXXX
XXXXXXXXXXXX XXXXXXX XXXXXXX
XXXXXXXXXXXX XXXXXXX XXXXXXX
XXXXXXXXXXXX XXXXXXX XXXXXXX
XXXXXXXXXXXX XXXX XXXXXXX
XXXXXXXXXXXX X XXXXXXX
XXXXXXXXXXXX XXXXXXX
XXXXXXXXXXXX XXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Turn 2: Player 0: "ian" playing @
Car status: Gear state: "Gear2"
Position vector (row, column): "Vector2D(10,32)"
Direction vector (x, y): "Vector2D(0,2)"
Applicable direction vectors (x, y): "Vector2D(-1,2), Vector2D(0,2), Vector2D(1,2)"
Lap status: (ian -> 0, m -> 0)
Please select a gear shift...
[W] ShiftUp - [S] ShiftDown - [0] NoShift
Build completed successfully in 3 sec, 605 ms (3 minutes ago)
```

[illegible]