Implementation of PECL 2

Summary

Basic version

The basic version of the project is made by the following scala classes:

- Parameters
- inputFunctions
- printingFunctions
- initializationFunctions
- aliensGenerator
- movementFunctions
- supportClasses
- advanceFunctions
- collateralEffectsFunctions
- reconversionFunctions
- Main

Parameters

The parameters class is an object class containing the settings of the program. This class was implemented to make the program more maintainable. If someone wants to change, for instance, the characters associated to the aliens or the probability of the commander generating a cloud, it can be done in this class, and it would affect the whole project.

inputFunctions

This class contains the functions used to handle the input of the user. The parameters inserted by the user can be incorrect and/or generate an exception. If this happens for the direction of the movement, the function returns an error message. If this happens for the other parameters, they are given a default value taken from the Parameters class.

printingFunctions

This class contains the functions used to print the stage matrix. There is only one public method called “printStage” which prints the stage matrix, the score, and the lives of the player.

This method calls the method “printMatrix” which prints one row of the matrix at a time by calling the “printRow” method. This last method prints a row of the matrix one character at a time and applies a color to it retrieving the value of the color from the parameters. After applying the color to the character, it resets the color of the output to its normal value.

initializationFunctions

This class contains the functions used to initialize the stage matrix. It has only one public method called “initializeStageMatrix”. This method runs through the matrix one row at a time and initializes it according to its position in the matrix.

To initialize the bottom row of the matrix, it calls the “initializeRowZero” method which sets to empty all the cells of the row but the one which corresponds to the initial position of the player.

To initialize the shield row, it calls the “initializeShieldRow”. This method uses a random number generator and a counter. The purpose of the counter is to count the number of consecutive shield cells in the row and make sure that there are at most 3 consecutive shields. If the cell is eligible for containing a shield, it uses a random number generator to determine if it will contain a shield or not.

To initialize the other rows, it calls the “initializeStageRow” method which simply sets to empty all the cells in the row.

aliensGenerator

This class inserts aliens in the top row of the matrix. It has one public method called “addAliens” which sets the head of the matrix to the row of aliens and concatenates it to the rest of the matrix using the tail attribute.

To generate a row of aliens, this class uses the function “generateAliens”. This function uses a random number generator to generate a number between 1 and 100, classifies it according to the values set in the Parameters class and generates an alien ship accordingly.

movementFunctions

This class contains the methods used to move the player in the bottom row of the matrix.

To move the player, we need to first find it. To do this, the class has a method called “findPlayer” which runs through the bottom row and returns the value of the column which contains the character associated to the player.

Once we find the position of player we need to determine if the movement is valid or not. To do so, the class has a method called “validMovement” which checks if the player is trying to move outside the matrix boundaries.

The class has a method called “playerMovement” which returns a new matrix with a new bottom row generated according to the movement of the player. This row is generated using the “applyMovement” function.

In case the player decides to use the automatic mode, there is a method called “automaticDirection” which generates a direction automatically. It first checks if the player is in one corner of the matrix and, if so, moves the player towards the center. Otherwise, it uses a random number generator to generate a number between 1 and 100, calculates its carry by 2 and uses it to generate a direction. By convention, we decided that 0 means left and 1 means right.

The last method in the class is “computeMovementDamage”. This method checks if the cell in which the player is moving is not empty. Since we are in the bottom row of the matrix, it is

reasonable to assume that if a cell does not contain the player and is not empty it contains an alien.

supportClasses

This scala file contains the classes used by the other scala classes exposed after this one. This file contains a class called “conversion_Element” which is used to store the entity and the position of the conversions. This file also contains the classes used to store the collateral effects of the destroyer and the cruiser. They are generalized using a sealed trait named “CE_object”, this way it is possible to create a list of generical collateral effects.

advanceFunctions

This scala class contains the functions to make the ships advance in the stage matrix, to compute the turn score, lives, and the advance damage.

The result of the advancement operation is computed as a new matrix generated row by row. The function that does it is named “advance”.

The row on top of the matrix is generated as an empty row because it will be filled with a row of aliens later.

The shield row is generated one character at a time considering that if a cell does not contain a shield, it will contain the content of the cell on top of it because the ships would advance normally. If a cell contains a shield, we need to consider that the aliens that collide with it will get destroyed and, in case they are cruisers or commanders, they will also destroy the shield.

The function that generates the row below the shield is made such that the shield is not moved but the other cells are.

The bottom row is generated such that the alien ships advance unless they would collide with the player.

All the other rows do not have anything special so we can make the content advance without conditions.

The function that computes the score of the turn scans the bottom row one character at a time and, if the analyzed character is different from the empty character or the player character, accounts the score of the analyzed alien ship. When the function finishes scanning the row, it returns zero in the base case.

The function that computes the turn lives scans the bottom row of the matrix looking for commanders and, for ach commander found, it accounts one life.

The function that computes the advancement damage checks if the cell on top of the player is not empty because, in such case, it is reasonable to assume that it contains an alien.

collateralEffectsFunctions

This file contains the functions to detect and activate the collateral effects.

Considering that the collateral effects can only be activated in the shield row and in the bottom row, the function that scans the matrix looking for the collateral effects to be activated in the turn only scans these two rows. Whenever a collateral effect is detected, it is added to a list which is returned as the output of the function.

This list will be used as the input of the function that activates the collateral effects. This function runs through the elements of the list and activates all the collateral effects contained in it. Every application of a collateral effect to the matrix generates another matrix which will be used as the input of the next recursion of the function.

The explosion of the destroyer is applied to the stage matrix as a matrix of positions. The cells affected by the explosion have the rows and columns in the following ranges:

$$row \in [explosionRow - EXPLOSION\ RADIUS, explosionRow + EXPLOSION\ RADIUS]$$

$$column \in [explosionColumn - EXPLOSION\ RADIUS, explosionColumn + EXPLOSION\ RADIUS]$$

For this reason, the function that applies the collateral effect of the destroyer runs through the rows of the matrix and if they are not in the explosion they are returned as they are. If they are involved in the explosion they get changed by the function “destroyer_explosion_row”. This function sets to empty all the cells whose column falls in the range of the columns affected by the explosion but the one containing the player which is left as is.

The collateral effect of the cruiser can affect either a row or a column so the function which activates the explosion must distinguish the two cases.

If the explosion affects the row, the function which applies the explosion to the matrix processes it. If a row is not affected by the explosion it is returned as is. If a row is affected by the explosion all its cells are set to empty but the one containing the player.

If the explosion affects the column, the matrix is scanned row by row and in each row only the cell in the affected column is set to empty unless it contains the player.

Finally, the functions to detect if the player has received damage work using the position of the player. There is no need to scan the whole matrix. In case of the explosion of the destroyer we can say that, considering the radius of the explosion and the fact that if an explosion occurs in correspondence of the shield, it affects the bottom row, it is safe to assume that the bottom row is always involved in the explosion. For this reason, we can determine if the player receives damage by checking if the column in which there is the player falls in the range of the columns affected by the explosion.

In case of cruiser explosion, we need to consider that it affects the player if either it affects a column and the column contains the player, or it affects a row, and the explosion occurs at the bottom of the matrix.

reconversionFunctions

This file contains the functions that apply the reconversions to the matrix.

First, we need to detect the reconversions by scanning the matrix cell by cell and checking if a cell is eligible for reconversion. If it is, we add it to the list of reconversions of the turn. This list will then be used by the functions that apply the reconversion.

The application of the reconversions starts by the scanning of the list and for each reconversion we generate a new matrix. The reconversions affect the element on the sides of the central one which means that they affect three rows. For this reason, we have a different function for each case. If a row is not affected by a reconversion is returned as is.

A similar reasoning is applied to the columns. The functions that apply the reconversions to the rows only affect the columns involved in the reconversions.

Main

This is the class that runs the game. It contains the instantiation of the objects containing the functions of the other files and it contains two functions to run the game. These two functions are “main” and “turnFunction”.

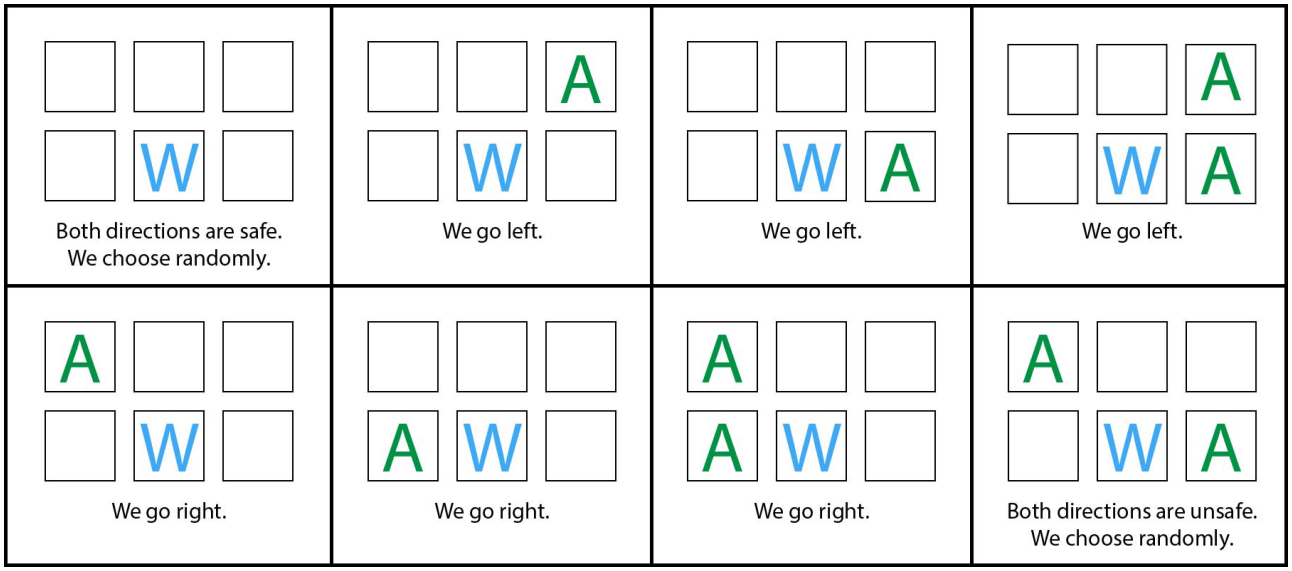
The main function first makes the user input the parameters of the game, then initializes the stage matrix and then starts the game calling the function “turnFunction”.

The function “turnFunction” shows the stage matrix to the user, asks for the direction of the movement, and then computes the stage matrix of the next turn and the other variables of the game.

Intermediate version

The intermediate version of the project has an improved function for the automatic movement. This function is made such that the player would avoid moving in a direction that would cause them to receive damage if it is possible to do so.

To understand how it works let us look at the following picture:



First, we need to consider that if the player is in a corner of the matrix, we must make it move towards the center.

If the player can move left or right, we need to establish if the directions are safe or unsafe. To do so we need to consider that the player will receive damage if either they move on top of an alien or they move under an alien because, in such case, they will receive advancement damage shortly after. For this reason, we need to consider the last row and the second last row as equally important when establishing if a direction is safe or not.

If the cells on a side of the player are both empty the direction is safe, otherwise it is not.

Let us see the following table:

leftSafe	rightSafe	Action
No	No	We move randomly
No	Yes	We move right
Yes	No	We move left
Yes	Yes	We move randomly

From the table we can see that if $leftSafe == rightSafe$ we need to generate a random number and use it to generate a direction. Otherwise, we can simply choose the safe direction.

Cloud version

Client

The cloud version of the project has another file called “networkFunctions” which contains a function that uploads the data to the Firestore database. This function receives the data to be uploaded in input from the main function, inserts it into a JSON object and sends it to the writing function of the server. To perform these steps, the function uses a library to make the JSON object and another library to make the web request.

To be able to import additional libraries in the project we had to change the build system of the project to “sbt” instead of “IntelliJ” and we had to change the build file called “build.sbt” adding the following importing statements:

```
libraryDependencies += "com.lihaoyi" %% "requests" % "0.8.0"

libraryDependencies += "com.lihaoyi" %% "ujson" % "3.3.0"
```

To be able to upload the score of the game we must slightly change the “turnFunction” function to make it return the final score of the game. So, in the base case of the function, we return the score after showing the “GAME OVER” message.

The values to be uploaded to the server are:

- The number of rows of the matrix
- The number of columns of the matrix
- The game mode
- The duration of the game in seconds
- The score of the game
- The name of the player

The additional values we need to collect are the duration of the game and the name of the player.

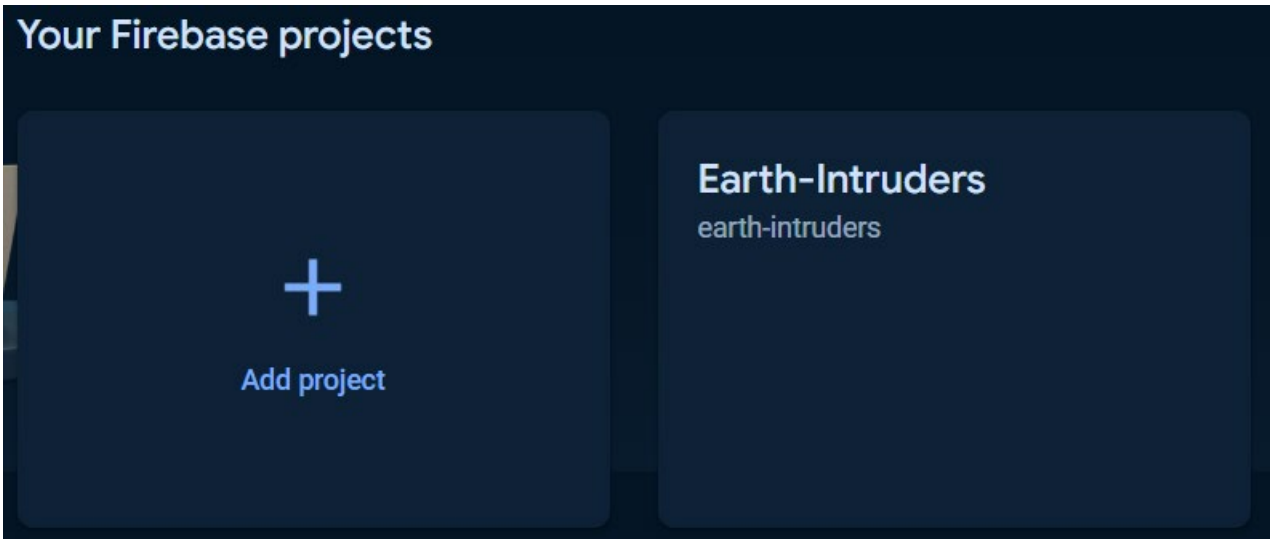
To get the duration of the game we need to save the timestamp of the instant right before staring the game and the one of the instant right after ending it. Calculate the difference of the values and convert it to seconds.

To get the name of the player we need to add a function to the file “inputFunctions”. The function to make the user insert the name of the player returns the default name in case the user does not insert anything. The default name is “DEFAULT” and is defined in the parameters of the program.

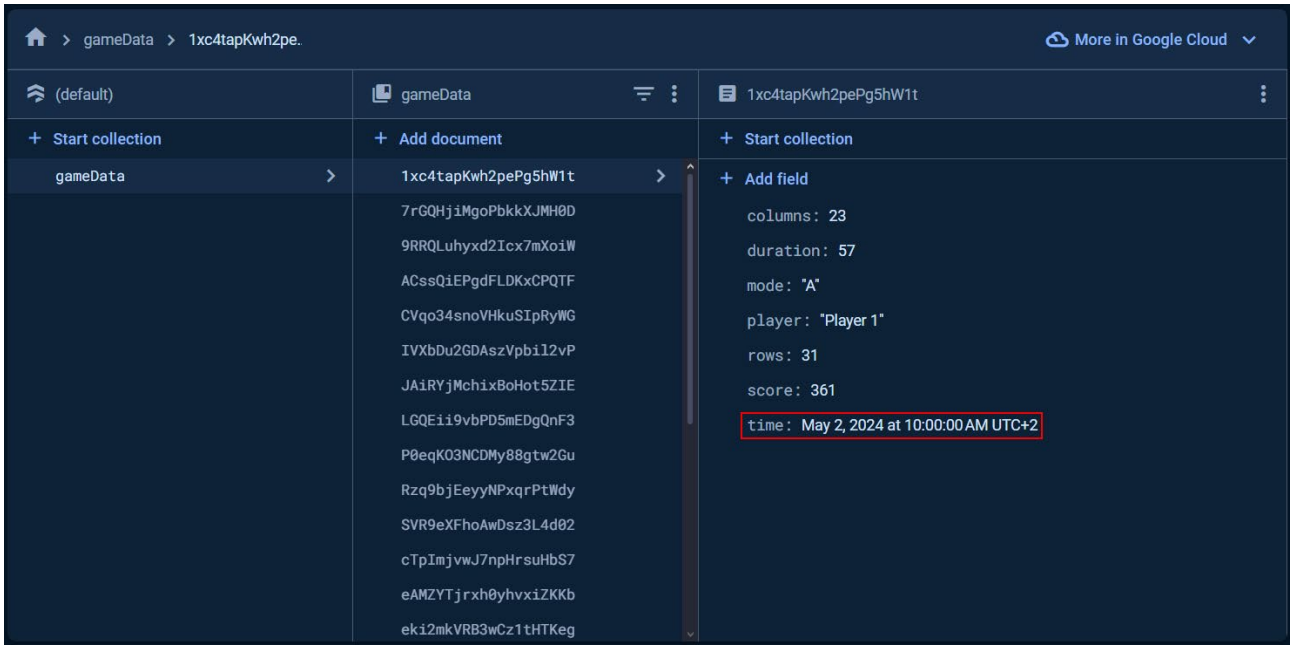
Server

The server of the project is a Firebase project which uses a Firestore database and two web app functions: one to read the data from the database and one to write it.

To create the server of the project we need to first create a web application is Firebase. We call it “Earth Intruders”.



Inside the Firebase project we create a Firestore database with one collection that we call “gameData”.



The documents in the collection have a field that we did not upload using the client. This is the timestamp of the game that we will use to sort the games.

The reason why we do not generate this value in the client is that Firestore uses its own particular timestamp format and provides an API to generate the timestamps. For this reason,

we just upload the rest of the data and we delegate to the server the task of generating the timestamp and inserting it into the database.

To create the two functions to read and write the data we need to first install node js and npm. After we do it, we must create an empty folder in the device and open a command prompt in it. In the command prompt we must use the following commands to start developing our application:

1. npm init -y: This instruction will initialize an npm project inside the folder.
2. npm install @google-cloud/firestore firebase firebase-admin firebase-functions: This instruction will install the dependencies of the application and update the package.json file.
3. firebase login: This instruction will open a browser window and make us login using a Google account.
4. firebase init: This instruction will initialize a firebase project inside the folder and ask us what kind of project we want to develop. We choose to develop functions and complete the rest of the procedure.

Now we can start developing our functions. We can do so in the index.js file inside the functions folder that has been created.

In the beginning of the file, we must import the libraries used by the application and retrieve the reference to the database using one of the imported APIs.

Now we can start writing the functions.

The function to read the content of the database is simply called “read”. In this function we first sort the data by the timestamp, and then we retrieve the first 10 documents of the collection. We use this data to generate the HTML content of a table and use this content to populate an HTML string with data. In the end we send the HTML string as the output of the response of the server.

As for the writing function we need to consider that the request of the client contains the data already in JSON format. In this function we get the data of the client by accessing the body of the request. Considering that it is a JSON object we can insert an additional field using the dot notation and we use this notion to insert the timestamp of the request into the data. Then we insert the data into the database, and we send a confirmation message to the client.

Now that we have completed the server functions, we need to deploy them to firebase opening the command prompt and using the command:

5. firebase deploy

At the end of the deployment process, the firebase server will provide us the links of the functions.

All we need to do now is to copy the address of the writing function into the client parameters and the project is completed!

Gabriele Colapinto