Project Document

Personal information

Project: Scale Game 23.4.2019
Name: Pekka Kosama
Student Number: 647764
Degree Programme: Bioinformation Technology 2nd year

General description

I have made a basic scale game with graphics, where one stacks weights on a set of scales to get a good score. There can be multiple layers of scales which work in tandem. How much “force” a weight puts on the scale depends on how far away from the centre it is, as in if it’s 4 slots away, it causes as much “force” as 4 weights a single slot away. The maximum amount of allowed imbalance is equal to a single weight placed on the end of the scale. Score is calculated according to the weight it causes to the scale, and if the scale is on top of another scale, score is multiplied by the distance of the slot of the scale that it is on.

Graphical interface is implemented with sprite based graphics. The players can be seen on the right side with their score. Player turn is indicated by green colour. Computer controlled players ended up not being implemented.

The game is started just by running a scala interpreter on the ScaleGameApp object inside the source code of the project. Game is controlled solely by mouse, the game tracks mouse movement and indicates the slot the mouse is over on and a player may add a weight by clicking on the slot.

Program structure

Program is built into several parts. The main app is the ScaleGameApp which launches and plays the game. It uses several different classes to make the game. Main class for it is the ScaleGame class which is the basis for the game. It initializes the game grid which works as the basis for the graphics etc. The grid is a 2-dimensional array inside the game class, which uses Locations as the base. Location objects hold information about what Item is in each of them. Location mostly has uses to changing the items on the location and returning each item, or the scale the location is on top of.

ScaleGameApp has a lot of methods for building the UI. There is a function for reading the scale information out of a .csv file. The function then adds the scales and the slots to the game. There are several different UI functions which update the game view and the game state to the players.

The Item is an abstract class which has 2 subclasses. Weight and Scale. Weight and Scale have some common methods, like giveScore, which helps calculate the score. Weight is what each slot of the Scale has if there is not a Scale. Weight works as holder for information about how much torque it causes to the scale and who “owns” the weight. Scale uses this information to calculate the balance and the score. Scale has also methods

for resetting the weight situation, checking the balance and updating the balance. The scales can cause a chain reaction of tipping if other scales are unbalanced.

The Item class isn't probably needed, and the weights and the scales could've been implemented separately. It works for calculating the score, but it could easily be implemented even without a superclass.

Player class is needed to keep the score of each player and player name and such. Player class would've had the AI as well, but due to time I didn't have time to implement it.

Algorithms

The most important algorithms and most complex algorithms used are in building the UI, for playerActions and for calculating the balance and score.

In the balance calculations, the algorithm goes over each slot and their weight, and calculates how much torque each slot causes and sums it all together. The function only calculates balance, it doesn't reset the weight. checkTip is used to check if the scale is not balanced, and the resetWeight method goes through all the slots and resets their weight. Because each slot might have either a Scale or a Weight, this is a method for both of those.

giveScore function works as a calculating tool for the score. It calculates the score for the given player by starting on the base scale. It goes through each slot and checks if a slot holds a scale or a weight. The function works recursively, as in if a slot has a scale, it requests it's score and multiplies it by the slot it is in.

The playerAction function works as a checking tool if the action of the player causes tipping on the scale, and if the action causes tipping of other scales. If also cycles through the players.

The ScaleGameApp has algorithms for checking which slot the player clicked on and then doing the action if the slot holds a Weight. It also has a function for making the spriteMap and updating it accordingly. It goes through each slot and draws the corresponding symbol. Empty slots are ".", weights have a number according to the amount of weights, and scales are T. Each empty slot under a weight has an "_" to make the game look a bit more "authentic". Each sprite can easily be changed in the future for better ones.

Data structures

There are several different types of lists, vectors and arrays. Most of the structures are immutable, because most arrays and vectors store classes inside of them. Therefore, there is no reason to use mutable lists. Some mutable arrays are used to store information until changed to a immutable. The base grid of the game is in an array that is not mutable, because all the locations stay the same. All the scales are stored in a Vector. There isn't a clear logical reason for each type of collection, they could be optimised according to their use, but it is a small performance difference in this level of computing. In the future, more careful planning of each of the collections would be beneficial.

Inside the classes, most of the information is stored in Options, because a lot of different things are easier with pattern matching. It made making the UI and different things a lot less convoluted.

Files

There are 2 picture files as png used for the program. The spritemap which consists of 36x44 sprites and could be easily modified to look different. It consists of all the letters, numbers and common other characters. There is a picture for the selection mask of mouseover.

There is also a scales.csv file for making the Scales. The format and the program don't check for some incorrect configurations, so the file must be constructed in a specific way. The csv has information on a single scale like this:

ScaleOn,2,2,1,4

ScaleOn: a string just for giving information. Disregarded in the code.

Numbers in order:

First one is for the level of the scale. It's mostly so that the program can draw it correctly.

Second one is for the number of the scale the scale is on top of. This can't be bigger than the existing amount of scales there is. The numbers start from 1, not 0 as usually in programming.

Third one is for the offset of the scale, which should only range according to the radius of the scale it is on.

The fourth one is for the radius of the scale. If it's too big, it might go over the limits of the game. There is error handling for all the basic cases. There is no error handling for super specific cases, but most cases are handled.

Testing

The testing process was completely different than the planned one. Most of the testing was done after the UI was done and it was done mechanically. Different cases of unbalance have been tested, the resetting and the scoring mechanics are all working correctly. The error handling for the files was also tested, so it shouldn't accept glaringly bad files. No unit tests were written.

Known bugs and missing features

Most obvious missing feature is the AI, which should have been implemented. There is no turn counter which causes the game to go on forever. There is no functionality for giving a .csv file of your own, you need to edit the existing one (or change the code to the new one). Input for player names isn't implemented.

There aren't any big bugs (that I know of). The basic functionality of the game should be bug free.

Best sides and weaknesses

The two best sides I consider are:

The way the scales work with weights and score. I'm proud of the way I wrote the giveScore and updateBalance functions and they work nicely. Also, the logic behind using the slots. There is a bit of sameness in these 2 functions, which could be made more concise.

The basic UI. I'm proud that I made the UI work with mouseover and the tracking of the mouse. It's is heavily influenced by the GraphicsExample given during the course, but I'm proud of figuring out how to use spritemaps and how to draw the basic grid. Also making it work without glaring errors makes me proud.

Weaknesses I consider:

File handling. How the .csv files are handled is quite cryptic and could be written more clearly. It handles some errors but mostly it just is a quick solution to implement a basic system.

Disorder in the classes. Some classes have some unused values and useless functions which could be removed. There is a bit of extra in most places.

No unit testing. As previously said, I handled most of the testing just using the UI without any different tests. I think I might've gotten some good use out of the unit tests had I written them and used it to check for problems in the code instead of checking it manually.

Deviations from the plan, realized process and schedule

The basic schedule with times:

16.3. The project started. Just some basic stuff for the basic classes. ~3h

29.3. Some implementations to existing classes. ~3h

31.3. Added the game class, fixes to existing classes. ~4h

10.4. Started building the UI. Lots of trying out different stuff. Decided to not use swing but went back to it after studying it a bit more. ~6h

12.4. Had to start rebuilding the base behind the game as building the UI seemed challenging with the existing stuff. All the stuff needed to be rebuilt so a lot of time to work on. ~5h

18.4. Got the game draw the UI. There is a lot of errors and lots of work to be done. ~4h

19.4. Got the UI drawn correctly. ~3h

20.4. Implemented mouseover stuff. Started to build the player actions with the mouse clicks. ~4h

21.4. Implemented the adding of weights and the resetting of the scales. Small logic errors still. ~4h

22.4. First working implementation. Added scoring system and visualisation for the player scores. ~5h

23.4. Mostly cleaned up the code. Added commentary on the main functions. Started doing the project document. ~2h

23.4. Still added basic file reading functionality. ~2h

23.4 Project plan and small fixes. ~5h

Total time: about 48 hours.

The difference to the plan was because of time management issues. Should've started doing the UI way earlier in the process and a lot of time scrunching ended up in the end. Most of the time was consumed during the fixing process. Order of progress was almost the same as planned, except the AI implementation was left out. More time was used on the basic stuff, and the total time used was less than planned (around 10-20h).

Final evaluation

All in all, I'm proud of what I've made. I hope I had implemented stuff I was supposed to such as AI and turn counters, but other aspects of the game are working well. I'm happy I have a somewhat finished product, because this was a serious test on my planning and time management skills (or lack thereof).

I'm especially proud of the UI and the basic functionality of the game. I had not made graphical UI before and I was happy that I got it work as well as it did. I like the way I implemented the functionality in the scale methods and how all of it works together.

I wish I had time to hone it to the best of my ability. There is a lot of things which could be improved, like removing some redundancy in the functions and variables. ScaleGameApp is quite bloated and could be made more concise with more information put into different classes. I should've tested the game more along the process, that way I could have prevented some of the roadblocks I ran into.

I'm happy that I made the structure of the program to be easily modifiable and continued. The base is now working, which lends to the continued modification of the program. New features could be easily implemented, like adding UI elements for getting the player info etc.

I would change the order I worked on the project. I would have started with the UI because it was the biggest difficulty during the project and ended up having to rewrite a lot of code because of it. I would also follow the original planning more closely, because doing it a bit free-form made evaluation of the stuff that needed to be done a bit difficult. This lead to having a lot of stuff to do in the final weeks.

References

The course material (especially the GraphicsExamples).

Scala documentation on swing.

https://www.scala-lang.org/api/2.9.2/scala/swing/package.html

Java documentation on awt:

https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

The spritemap:

https://itch.io/jam/lowrezjam2016/topic/19413/minimal-sprite-font-with-upperlower-cases-cleanreadable

Pictures of the game in play: