General plan

Personal information

Project: Scale Game 15.2.2019

Name: Pekka Kosama

Student Number: 647764

Degree Programme: Bioinformation Technology 2nd year

General description

I'm going to do the Scale Game, which is a balance game where one tries to stack weights on scales to get a good score.

In the scale game there will be scales on which the player puts weights on. There may be either a single scale, or a single scale on the bottom with other scales on top of it. Weights apply more torque if they're further from the middle point of the weight. The maximum amount of allowed imbalance is equal to a single weight placed on the end of the scale. The game will score the weights based on the "weight" they apply on the bottom scale. The game will include a graphical interface and possibility for computer controlled opponents. If a player puts a weight on top of another players weight, he will "own" the weights under it.

I'm planning on implementing the project with a graphical user interface and an intelligent computer opponent.

Draft user interface

There is going to be a graphical user interface. In the program there will be a main window with the game. The main window will be grid-based, so the scales can be easily built with picture files. The scales will be built with sprite-based picture files, so each square in the grid hosts one of the sprites. In the main window, you can see the scales and the current amount of weights left and which player's turn it is. Score

could also be seen (especially in the earlier versions for easier testing), but in the complete game I will probably remove it (so it will be more gamey). Different players' weights are colour-coded. After all the weights are set, the program will tell the final score and which player won in a pop-up window.

There will be a starting window which asks how many players are playing and how many of them are computer controlled. The game will also ask for different scale setups (eg. 1 big scale, 1 big scale with 2 smallers scales, etc.). The game will be controlled with a mouse, and with a mouse click a player may place a weight on the scales.

You can press esc any time during the game which will prompt a pop-up window which will ask if the player wants to exit the game.

Files and file formats

There won't be need for many files. Most of the graphics will be sprite based with image files like jpegs. I will make some preset scale sets so I can test the game and the AI better, but mostly the scale sets will be randomized in the final product. The scale sets are saved in human-readable-IO text files. The format is described after the next paragraph with weights included.

There will also be preset gamestate files which will include already dropped weights to test the AI. They will be written in the same format as the preset scale sets, but with some weights applied to them:

BaseScale: (the width of the scale)

Loc: (-width to width) item:(what on it, e.g. Scale1 or weights: (number of weights): (player symbol(e.g. a or b)))

Loc: 2 item: weights: 2:a

Scale1: (the width)

(item locs like before)

Scale2: 3

etc.

Plan for system testing

System testing will be implemented in phases along the implementation of different features. The most important parts to test are the basic functionality of the scales (tipping, calculating weight), scoring, setting up the game board and putting weights on the board including checking the correct player. In the later stages of implementation, testing that the AI works as intended (if partly randomized, with a set rng-seed) and that the game reads the premade game boards correctly (so that the AI can be tested).

Scales should be tested on different types of setups (single scale, multiple scales etc.), tested by putting the weights on the board and trying to tip the scales with different amounts of weights. Scoring should be tested similarly. Setting up the game board should be tested with the randomised setup with a set rng-seed and with the premade files. Also the function of switching the owner of a weight should be tested.

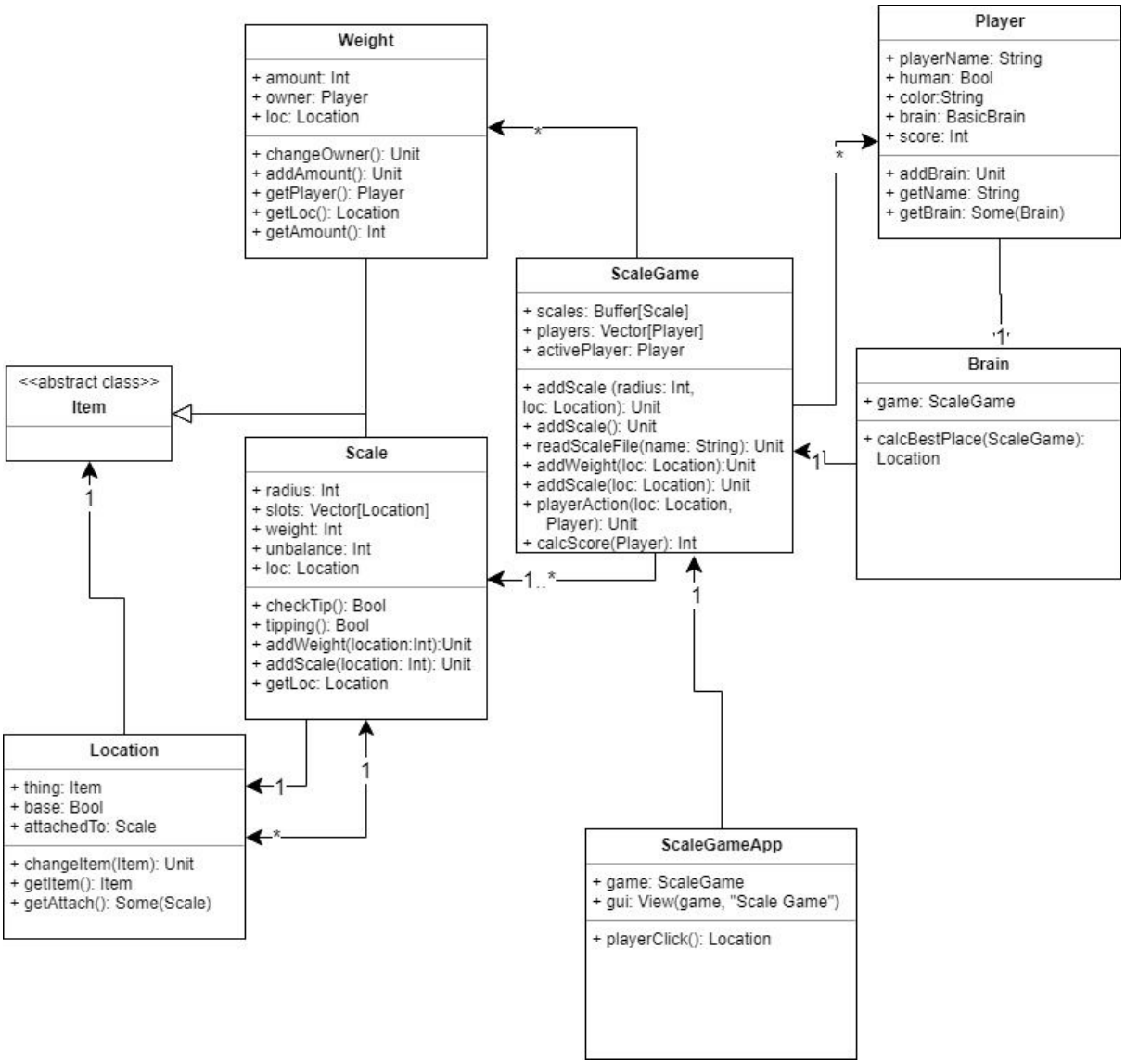
AI should be tested with different kinds of setups. AI should probably only be tested with one or two moves, so it would be easier to plan the different situations. Especially if the plan is to optimise the AI, I think this could be a good way to go with the testing.

The control inputs will be tested as well, with simulating mouse clicks on a certain part of the area with different scale set ups and testing for the esc-command.

There will also be error handling for the files.

Technical plan

Class structure



The program will be built on a small amount of classes. There is the ScaleGame class which is the basis for the program. It will initialise the scales and different locations for the game. It also contains information of the Players, Scales and the active player. It also has functions built in it to read the scale file and do a playerAction.

Location class is used as a base to locate the Weights and Scales. It will also help to locate different Items on the game board. Under the item abstract class is the Scale class which the game will mostly be built around. It will have all the Locations in

which a Weight or another Scale can be put on. It will hold information about it's radius, the Locations on it in a Vector, weight on it, and the imbalance status. The Weights have information how many weights are on the location. This number can be zero, so every Location has either the Weight (so you can put weights on it) or a Scale. Every Weight has an owner.

The Player class contains the information on the player. A Player may be either human or computer controlled. Each player has a color to indicate the ownership of the weights. The computer controlled players have an Brain class to build the AI on. I will code more Brain classes if I have time at the end of the project.

There will also be the ScaleGameApp which is the launchable app. It will build the gui based on the game board. I haven't figured out if I need to build different classes for different UI elements or if I should just build it in the app. This will be implemented later in the project.

Use case description

The game begins with a pop-up window asking for different settings. There will be the amount of scales, amount of weights to put, how big the base scale is. There is going to be a maximum of 4 players. The user will input player names and if the player is computer controlled. Then the game will begin with a base scale according to the setting. The player will put down weights with mouse clicks to get a better score. After each round of players, there another scale will be added to the game, randomised. If the scale is imbalanced, as in a side has more weight than it's radius would allow, the scale will tip and be lost. If the base scale is tipped, the game will end in a loss for the player who tipped it.

After all the weights have been set, the game will end and the score will be calculated according to the proportional weight on the base scale.

Algorithms

There will be a few algorithms in use. The most important one is calculating the score. It will use a recursive function with for loops to calculate all the score for each player.

There will also be algorithms to check the tipping of scales. For example if a scale tips, and it causes the next scale tip, there will be an algorithm to check the affected scales.

The most complicated algorithm will be the AI of the computer controlled players. I'm going to make challenging computer player, as in one that doesn't randomly choose the place to put the weight. It will also try to tip the scale if it would be advantageous for the player. If I have time to computer different kinds of AI, I will try to make them have differing personalities (e.g. will not tip scales) or differing difficulty (e.g. won't make the optimal play, or a random player). The algorithms will probably check the advantage of putting a weight on each Location and evaluate the score change and decide the final action based on that.

Data structures

There will a mutable list of scales in the ScaleGame class because of the possibility of scales tipping and adding new Scales. The list of players will be in a immutable vector because there is no need to change the players during the game. On the scale will be the list of Locations on the scale, which also will be in a Vector because the locations will stay the same and it would only cause problems if the list would be mutable.

Schedule

Weeks 9-10:

Base classes (ScaleGame, Item, Scale, Weight, Location): 8-10h
Tests for the base classes 3-5h
Player Class 2-3h

Weeks 11-12:

ScaleGameApp 5-7h (with graphics)
Tests for Player and ScaleGameApp 2-3h
Programming base AI 5-7h
Tests for AI 3-5h

Weeks 13-14:

Refining the Base classes 8-10h
Refining tests 3-5h
Refining ScaleGameApp 8-10h

Weeks 15-16:

Refining AI 8-10h
Refining AI tests 5-7h
Finishing touches 8-10h

Unit testing plan

I'm planning on testing all the major features of the program. The tests will be written alongside the classes. First the base classes, the scale functionality will be tested along with adding weights and adding scales on top of other scales. The tipping tests will be done with different situations. First going with the base scale, then going along with scales on top of each other. Also it will be tested that the tipping of scales works correctly, so that the location the scale was on will be changed to an "empty" Location (meaning that the Location will have a Weight with the amount 0.)

Then the weight calculations will be done as well, so that the scoring will be done correctly and the weight will be calculated to the bottom tiers correctly. The adding of new weights on top of a existing weight will be tested as well, and at the same time the changing of ownership.

In the location class, the switching of the Locations Item will be tested. The base Location is a special case, so the functionality of it will be tested as well.

In the ScaleGame each of the different functions will be tested. Especially important is to test the addScale functions extensively. The Location specific will be tested along with the readable files, and the not specified will be tested with a rng seed. Most important function to test is the calcScore, to check that it will return the correct amount of points in all cases. It will have several different tests because the extent of the different situations is massive.

Another important aspect to test is the AI. Basically the tests will check if the AI will make the choices as it is thought to be programmed to. If there will be different kinds of AI, they will be tested as well, accordingly.

References and links

I haven't used any material other than the course material so far. I will probably use some material for doing the graphics.